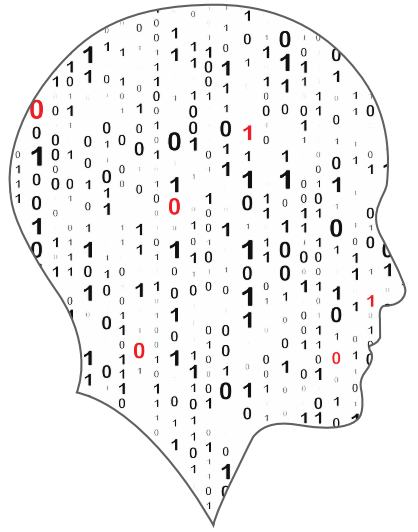
PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

An Roinn Oideachais agus Scileanna
Department of Education and Skills

National Workshop 6



LEAVING CERTIFICATE
COMPUTER SCIENCE

Schedule

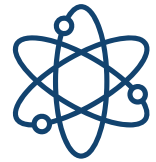
9.00am – 11.00am	***Session 1:*** **LCCS Higher Level Topics**
11.00am – 1.00pm	***Session 2:*** **Creating an Inclusive Classroom: SEN for Computer Science**
	Lunch
2.00pm – 4.30pm	***Session 3:*** **Post Phase 1 CPD** **Revision / Preparation for final Assessment**

Key Messages

LCCS can be effectively mediated through the use of a constructivist pedagogical orientation which will incorporate participatory and inquiry-based learning activities (whole-class, group, pair or individual).

LCCS aims to develop and foster the learner's creativity and problem solving skills along with their ability to work both independently and collaboratively.

Digital technologies used in LCCS have the potential to enhance collaboration, learning and reflection, by enabling students to learn more efficiently and to facilitate work that might not otherwise be possible.



The Turing Machine stands as the modern definition of computability.

Professional Development continues for Phase 1 teachers from September.

LCCS is suitable for all! This includes students with SEN and of all ability levels.

LCCS Higher-Level Topics

By the end of this session

Participants will be enabled to:

- Develop a deeper understanding of the time complexity and limits of algorithms
- Recognise situations when heuristics should and could be used as a method of problem solving
- Understand the significance of Turing Machines and how they operate
- Gain additional insights into higher-level topics such as:

  software development methodologies, relational databases, recursion, user-centred design, Artifical Intelligence (AI) and machine learning and communication protocols.
- Appreciate more fully the use of a variety of pedagogical approaches for Teaching, Learning and Assessment of the above

LCCS Learning Outcomes – Higher Level

1.10 discuss when heuristics should and could be used and explain the limitations of using heuristics

1.16 compare two different user interfaces and identify different design decisions that shape the user experience

2.9 assemble existing algorithms or create new ones that use functions (**including recursive**), procedures, and modules

2.10 explain the common measures of algorithmic efficiency using any algorithms studied

2.12 describe the different types of logic gates and **explain how they can be arranged into larger units to perform more complex tasks**

2.15 explain what is meant by the World Wide Web (WWW) and the Internet, including the client server model, hardware components **and communication protocols**

2.21 critically reflect on and identify limitations in completed code **and suggest possible improvements**

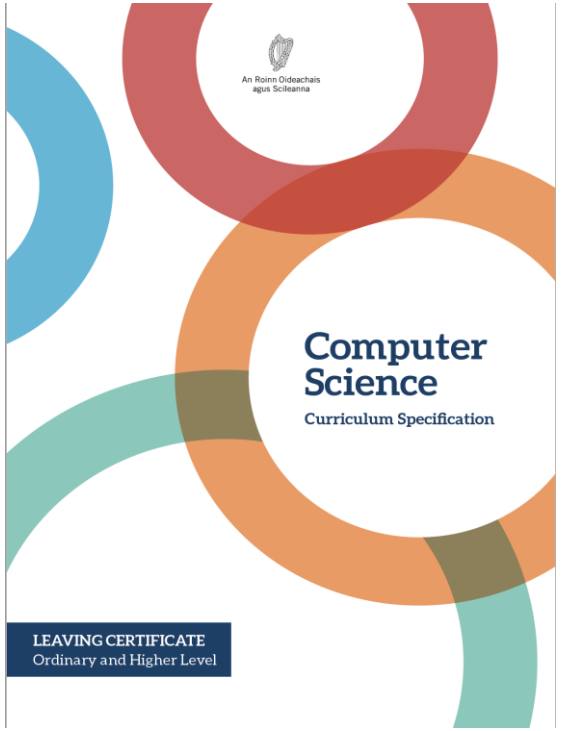
3.2 create a basic **relational** database to store and retrieve a variety of forms of data types

3.5 structure and transform **raw** data to prepare it for analysis

3.9 analyse and interpret the outcome of simulations **both before and after modifications have been made**

3.10 explain the benefits of using agent-based modelling and how it can be used to demonstrate emergent behaviours

Session Topics



Algorithmic Complexity (LO 2.10)

Heuristics (LO 1.10) Turing Machines

Artifical Intelligence / Machine Learning

Comparing User Interfaces (LO 1.16)

Relational Databases (LO 3.2) **Designing and Developing**

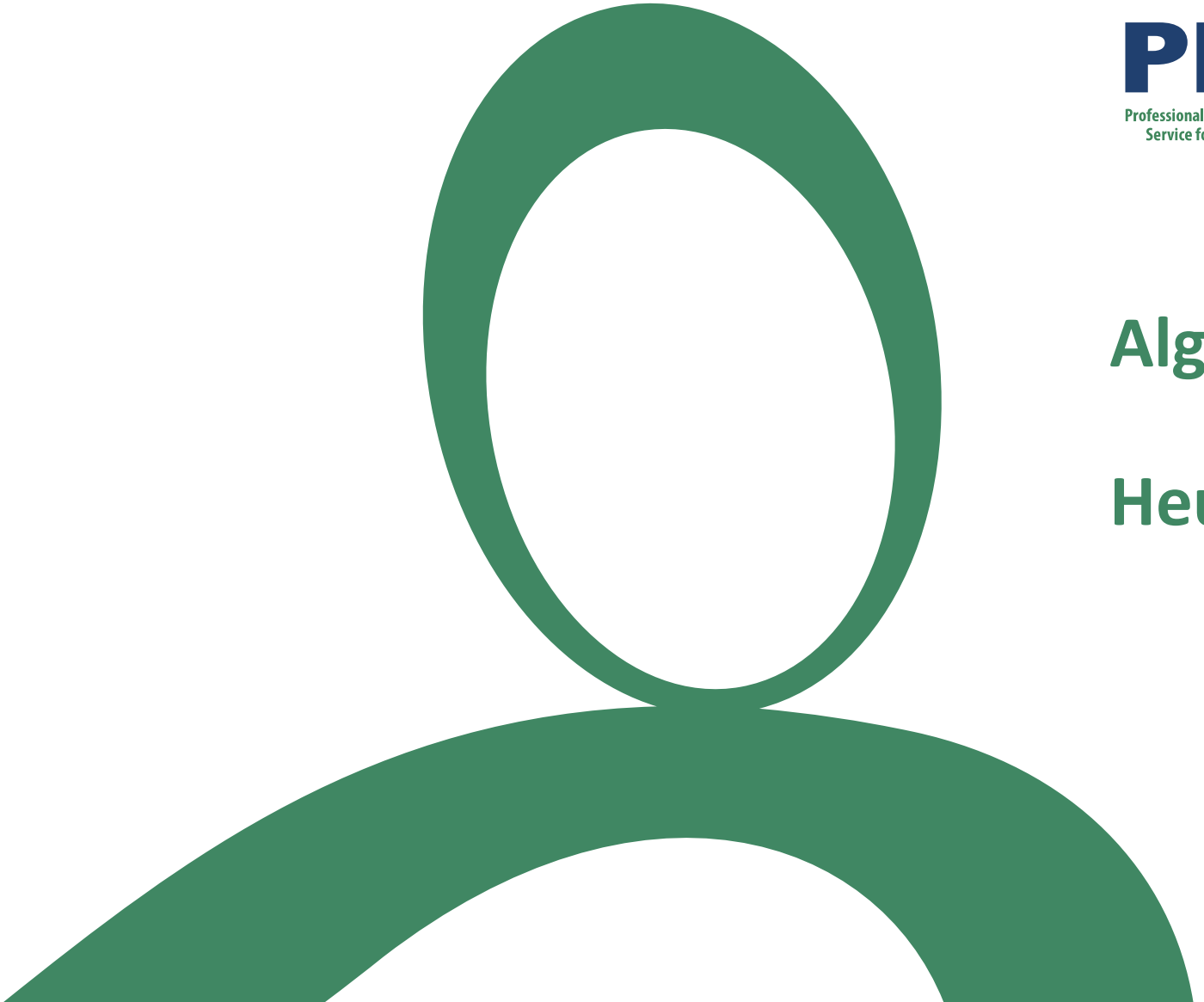
Raw data (LO 3.5)

Staged vs. Iterative (LO 1.19)

Roles and Responsibilities (LO 1.20)

Communication Protocols (LO 2.15)

Recursion (LO 2.9)



PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Algorithmic Complexity

Heuristics

Overview (The Big Picture)

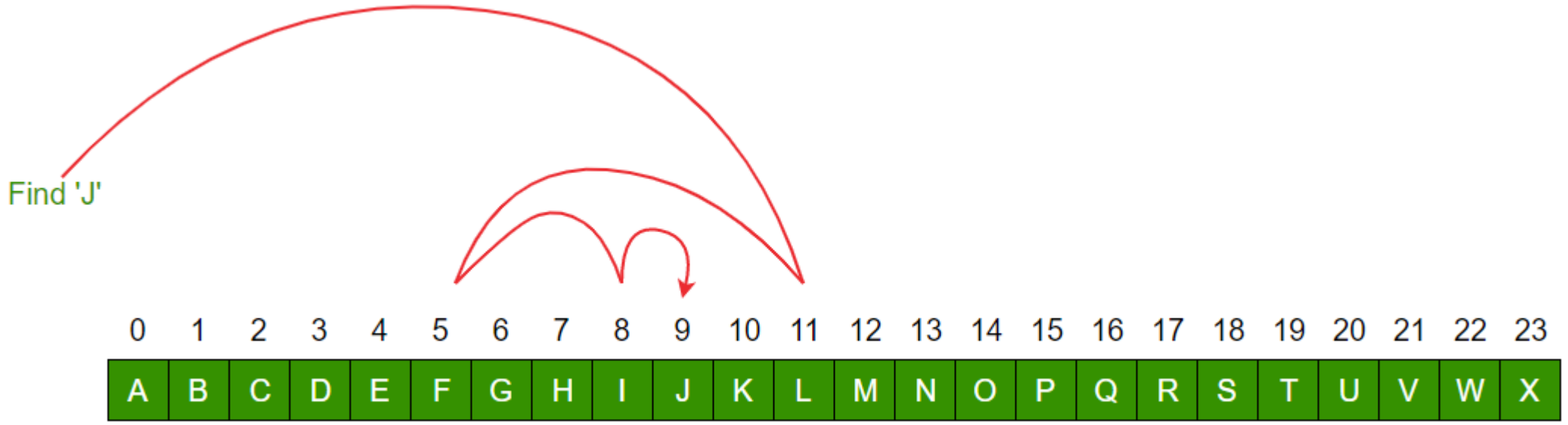
> "An algorithm is a set of rules for getting a specific output from a specific input. Each step must be so precisely defined that it can be translated into computer language and executed by machine"
>
> Donald Knuth (1977)

- **Algorithms are used to solve problems**
- **Complexity allows us to classify algorithms (as 'good', 'fair' or 'poor' in terms of performance) and therefore compare algorithms**
- **Problems can be classified as easy (polynomial, class P), hard (exponential, class NP) or unsolvable (impossible)**
- **Heuristics is an approach to solving 'hard' problems**
- **If a problem can be solved using a Turing Machine it is computable**
- **Problems for which there is no Turing Machine solution cannot be solved**

Algorithmic Complexity (Efficiency)

- **Algorithms have both space (memory) and time (CPU utilisation) requirements**
- We want a fair way to analyse algorithms (complexity analysis) – needs to be machine independent
- **The main concern is to find the worst case performance. Why?**
- How much time is required to find the smallest number in a sorted list? (1 operation => constant)
- **What if the list was unsorted? (potentially n operations => linear time complexity)**
- What about them amount of time it would take to find a specific element in a sorted list? (search space is halved on each comparison => logarithmic time complexity)
- **Elementary Sort algorithms have quadratic time complexity. This means that sorting 5 shelves of books will take 25 times longer than sorting a single shelf (not 5 times longer!)**

Binary Search Example

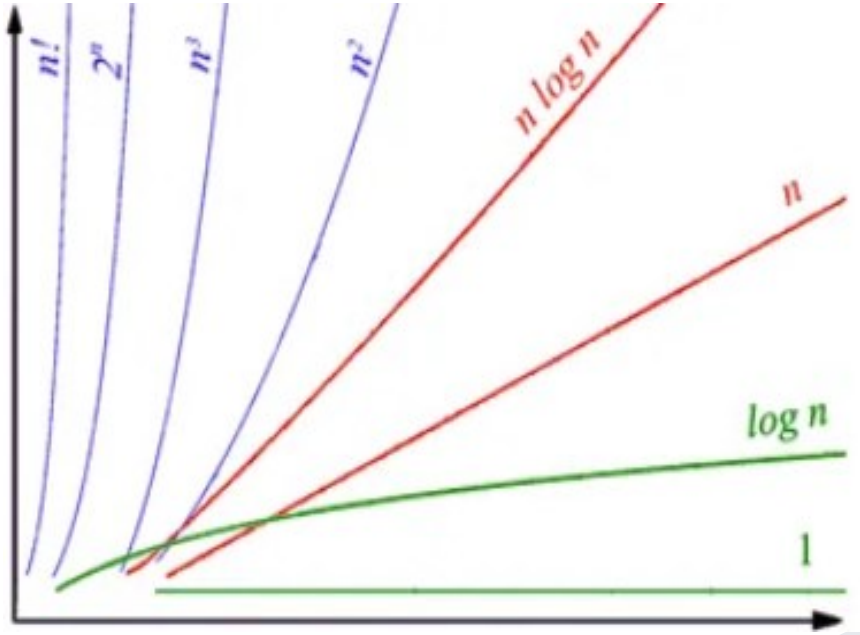


Source: geekforgeeks.com

Algorithmic Complexity (Efficiency)

Big-O notation provides a way to talk about the kind of relationship that holds between the size of the problem and the program running time. A shorthand notation for measuring worst case complexities. It is inexact by design.

O(1)	*Constant Complexity*
O(n)	*Linear Complexity*
O(n^2)	*Quadratic Complexity*
O($\log_2 n$)	*Logarithmic Complexity*
O($n \log_2 n$)	*Linearithmic Complexity*
O(2^n)	*Exponential Complexity*
O(n!)	*Factorial Complexity*



The number of operations (*y-axis)* versus input size *n*

Summary: Algorithmic Time Complexity

Programmers need to be aware of time complexity of the algorithms they write (or choose to use)

Algorithms that have constant, linear and polynomial time complexities (i.e. tractable algorithms) are all considered useful. Be careful with quadratics!

	Best Case	Average Case	Worst Case
Linear Search	$O(1)$	$O(n)$	$O(n)$
Binary Search	$O(1)$	$O(log_2 n)$	$O(log_2 n)$
Simple (selection) Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Quicksort	$O(n\ log_2 n)$	$O(n\ log_2 n)$	$O(n^2)$

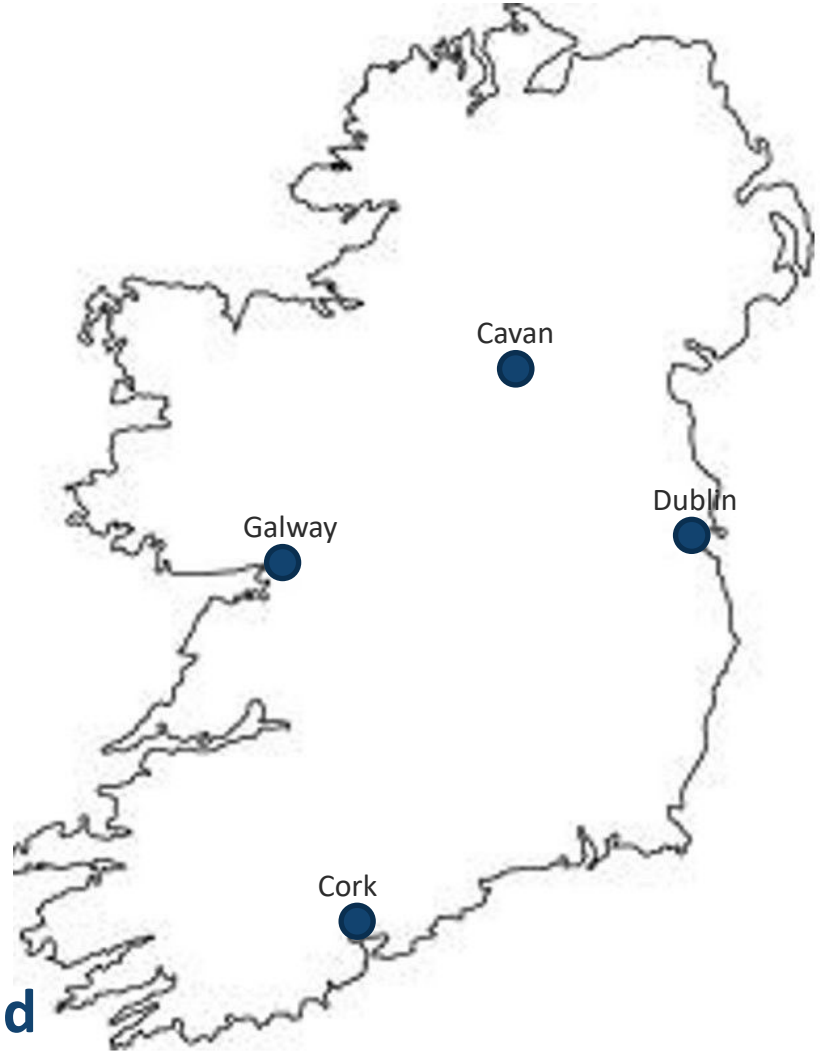
Algorithmic Complexity: Limits of Algorithms

Example: The Travelling Salesperson Problem (TSP)

Suppose you decide to ride a bicycle around Ireland

- **you will start in Cavan!**
- **the goal is to visit Dublin, Cork, and Galway before returning to Cavan**

	Cavan	Cork	Dublin	Galway
Cavan	-	304	115	168
Cork	304	-	258	206
Dublin	115	258	-	209
Galway	168	206	209	-



What is the best itinerary?

- **how can you minimise the number of kilometers and yet make sure you visit all the cities?**

Real World Applications

For many applications the number of “cities” (*n*) can be thousands or more.

While it is not likely anyone would want to plan a bike trip to thousands of cities the solution to finding the shortest tour of a large number of cities can be applied to (is the same as) many important “real world” problems:

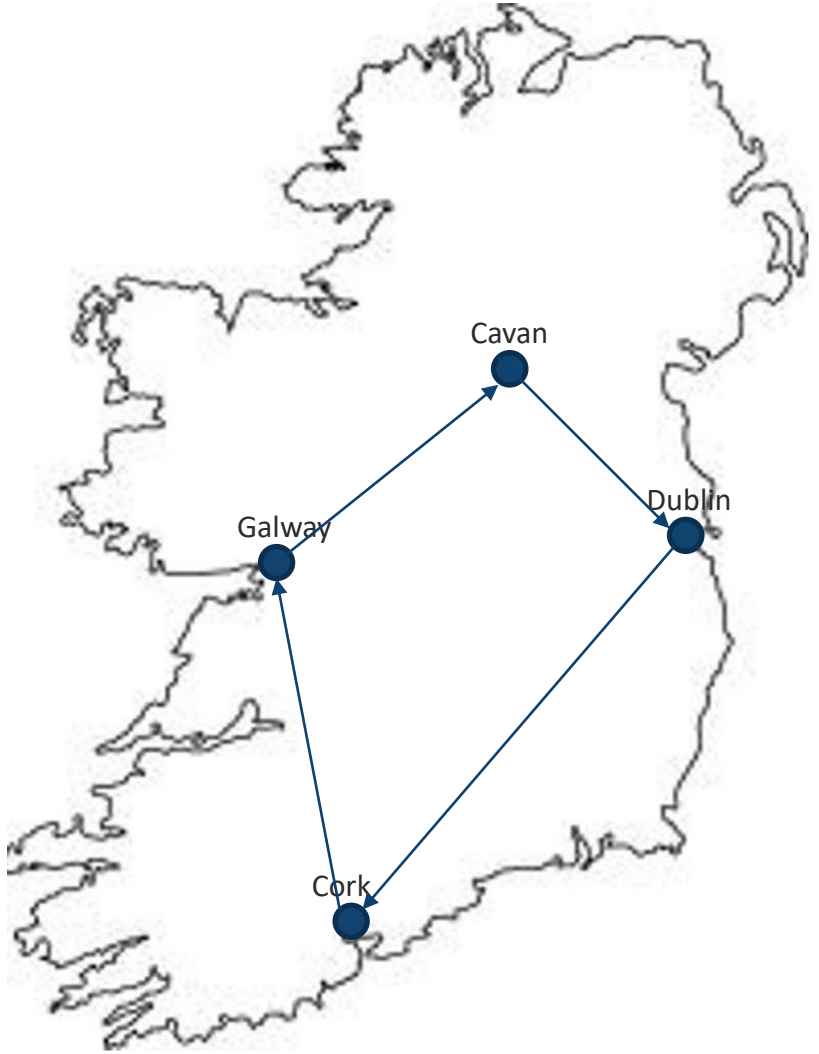
- transportation: school bus routes, service calls, delivering meals, post/parcel deliveries, delivery of online purchases (e.g. Amazon)
- manufacturing: an industrial robot that drills holes in printed circuit boards
- design: VLSI (microchip) layout
- communication: planning new telecommunication networks
- space exploration: minimise the use of fuel in targeting and imaging manoeuvres for the pair of satellites involved in NASA *Starlight* space interferometer program
- biology: to compute DNA sequences

Sources: Lero and Explorations in Computing (2012), John S. Conroy

Algorithmic Complexity: Limits of Algorithms

A Brute-Force approach would find all itineraries and then pick the best.

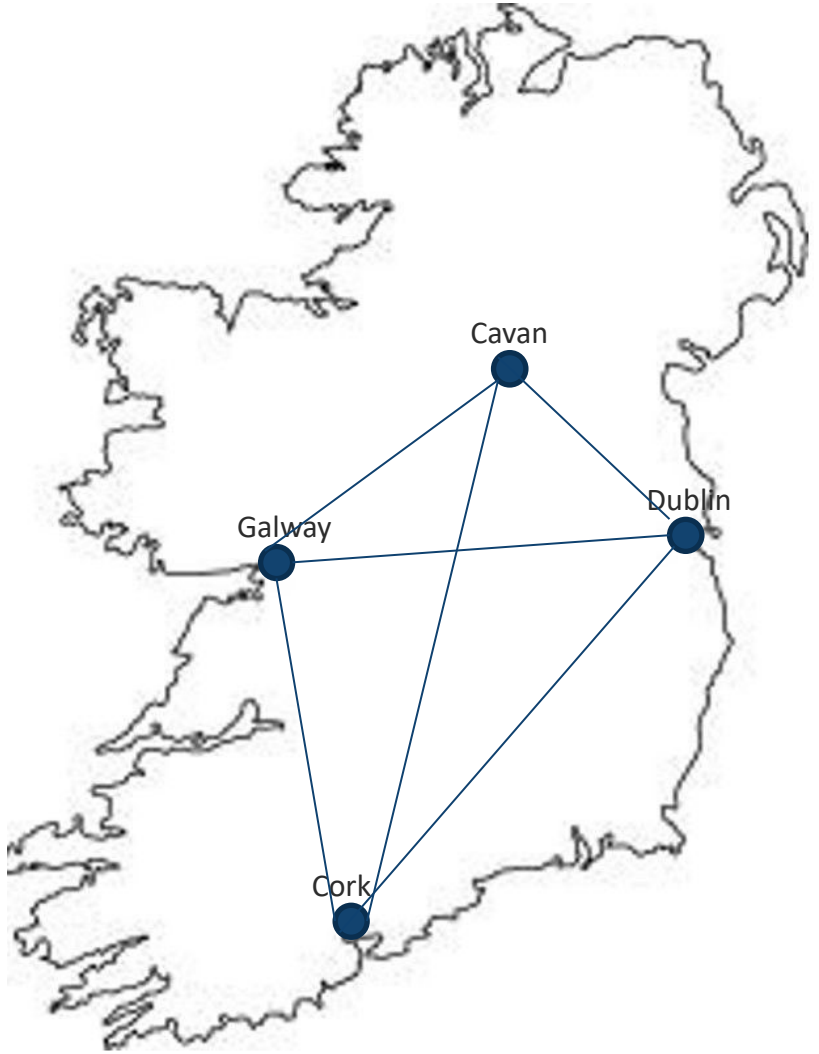
Cavan – Dublin – Cork – Galway – Cavan **747km**



Algorithmic Complexity: Limits of Algorithms

A Brute-Force approach would find all itineraries and then pick the best.

Cavan – Dublin – Cork – Galway – Cavan	747km
Cavan – Dublin – Galway – Cork – Cavan	834km
Cavan – Galway – Cork – Dublin – Cavan	747km
Cavan – Galway – Dublin – Cork – Cavan	939km
Cavan – Cork – Galway – Dublin – Cavan	834km
Cavan – Cork – Dublin – Galway – Cavan	939km



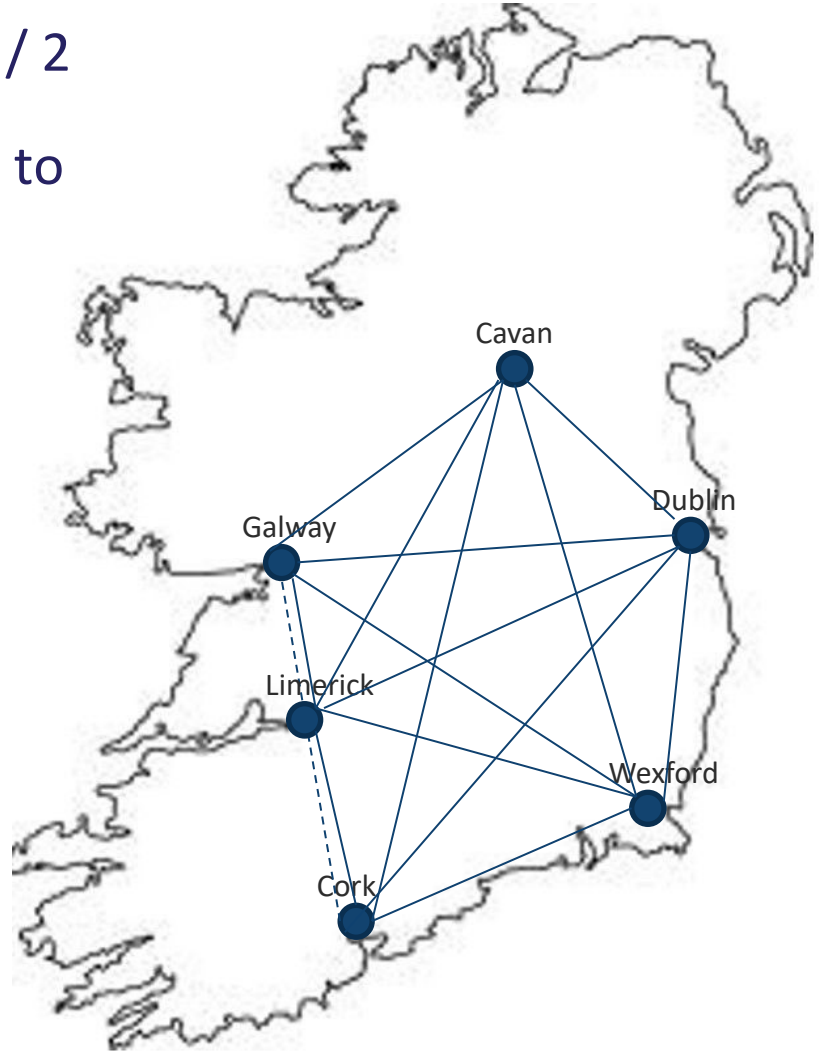
Observations? (Evaluation and Testing, Computational Thinking – recognising patterns)

Algorithmic Complexity: Limits of Algorithms

The number of possible tours of a map with n cities is $(n - 1)! / 2$

The number of tours grows incredibly quickly as we add cities to the map

#cities	#tours
5	12
6	60
7	360
8	2,520
9	20,160
10	181,440



The number of tours for 25 cities is 310,224,200,866,619,719,680,000

Edpuzzle Activity

Heuristics

- **An approach to problem solving.**
- Mental shortcuts that can be used to make a quick decision.
- **Heuristics are the strategies derived from previous experiences with similar problems.**
- Not guaranteed to be optimal (but usually take less time than would be required to find an optimal solution). Limitations lie in the trade-offs e.g. correctness vs. performance.
- **Suitable when finding an optimal solution is impractical or impossible** e.g. TSP heuristic may be to pick whatever is currently the best next step regardless of whether that prevents (or even makes impossible) good steps later (known as the greedy algorithm).
- Some common examples of heuristics include trial and error, a rule of thumb, an educated guess and intuitive judgement

Source: http://www.popflock.com/learn?s=Heuristic

Heuristics (ctd.)

When we cannot solve a problem exactly, one common approach is to use a heuristic instead. A heuristic is a type of algorithm that does not necessiarily give a correct answer, but tends to work well in practice.

The trade-off criteria for deciding whether to use a heuristic for solving a given problem:

- **Optimality**: When several solutions exist for a given problem, does the heuristic guarantee that the best solution will be found? Is it actually necessary to find the best solution?
- **Completeness**: When several solutions exist for a given problem, can the heuristic find them all? Do we actually need all solutions? Many heuristics are only meant to find one solution.
- **Accuracy and precision**: Can the heuristic provide a confidence interval for the purported solution? Is the error bar on the solution unreasonably large?
- **Execution time**: Is this the best known heuristic for solving this type of problem? Some heuristics converge faster than others. Some heuristics are only marginally quicker than classic methods.

Source: https://softjourn.com/blog/article/heuristic-programming

We now turn our attention to focus on a fundamental question of Computer Science:

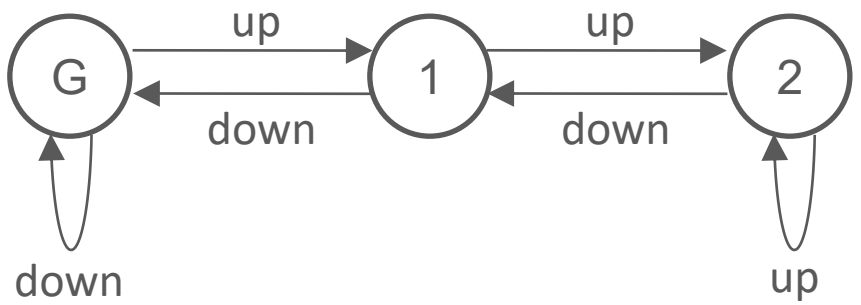
What is computable?

Turing Machines

https://www.turing.org.uk/

Turing Machines - Introduction

The illustration below is of an elevator represented as a finite-state machine



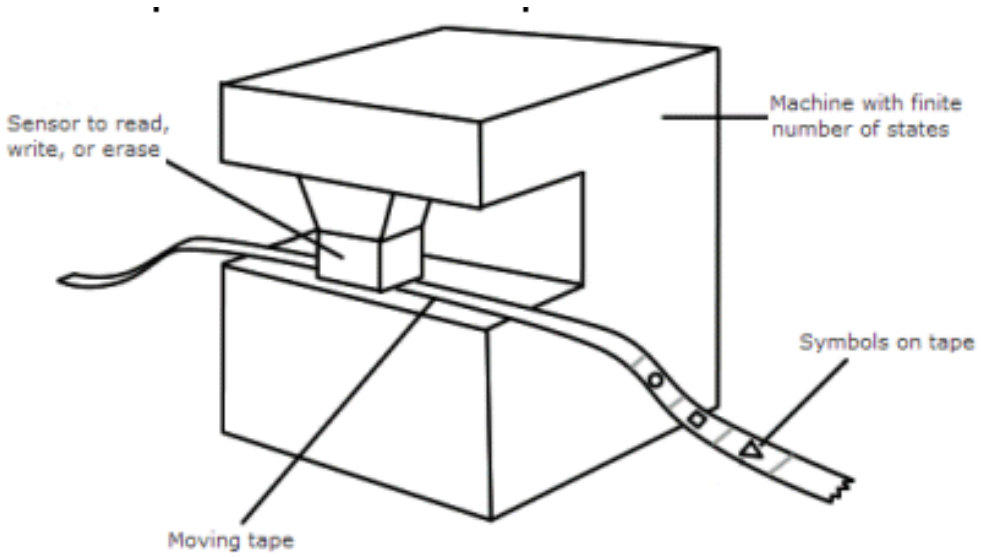
- Circles represent states (in this case floors)
- Arrows between circles represent transitions between states
- The labels on each transition represents the button press event

What happens when we are on the ground floor and press the **UP** button?
What happens when we are on the ground floor and press the **DOWN** button?

Turing Machines - Introduction

The Turing Machine (TM) was invented in 1936 by Alan Turing.
It is a basic abstract symbol manipulating device that can be used to simulate the logic of any computer that could possibly be constructed.

Although it was not actually constructed by Turing, its theory yielded many insights.



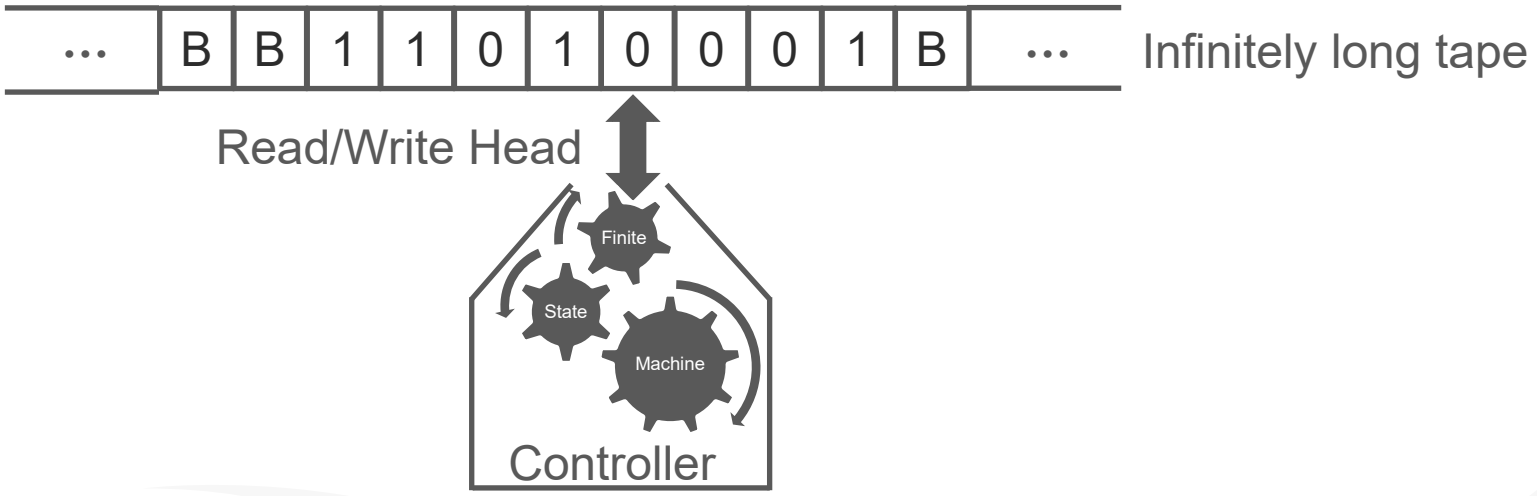
Turing Machines - Introduction

A Turing Machine consist of three components as follows:

1. An infinitely long tape made up of individual cells. Each cell can contain a single character – typically 1, 0, or B (blank)

2. A read/write head pointed at an individual cell

3. A controller (aka finite-state machine) which instructs the read/write head what to do



A schematic representation of a Turing Machine

Turing Machines - Operation

Initially the tape is inscribed with a sequence of characters – called the input

For example:

···	1	0	1	0	0	1	···

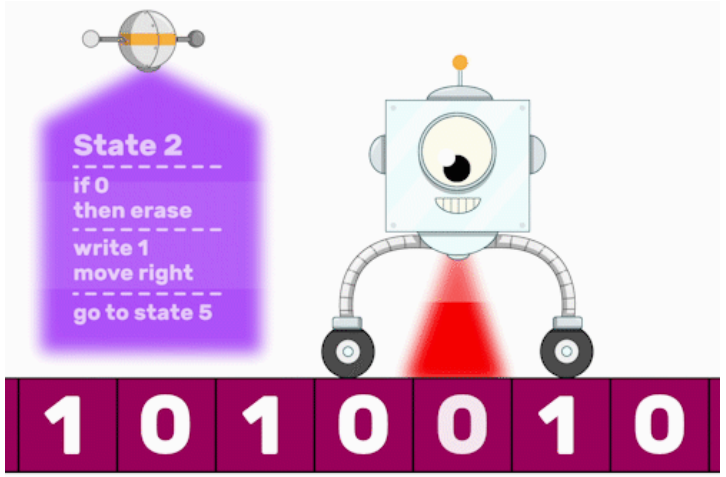
The operation of the Turing Machine is controlled by the finite-state machine (controller).

The operation takes place as a sequence of steps known as transitions

The controller decides for a given (input character, state) pair, the (output character, state) pair - know as a transition.

Each transition involves:

- **Reading**
- **Writing**
- **Moving**
- **Updating**

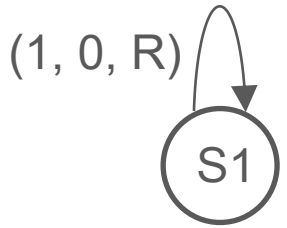


Turing Machines - Operation

Transitions can be expressed using:

state transition tables **OR** **state transition diagrams**

Current State	Read (input)	Write (output)	Direction to move r/w head	Next State
S1	1	0	Right	S1

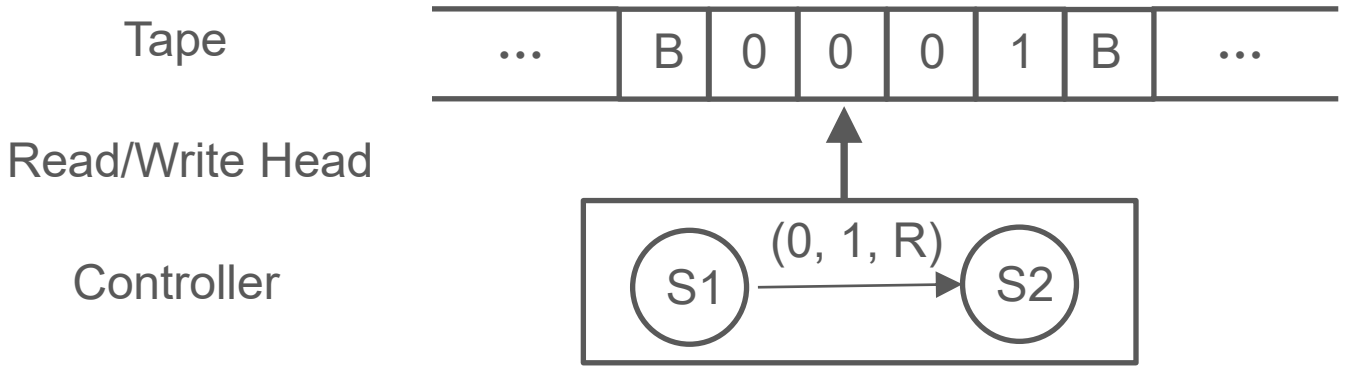


The above state transition table and diagram shows a single transition which says:

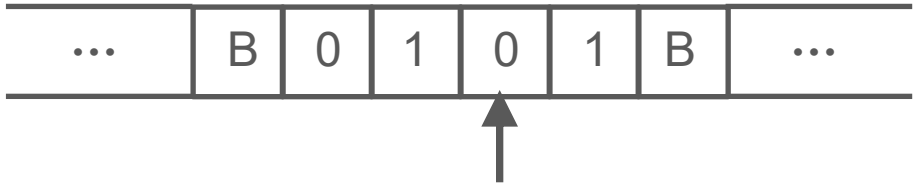
When in state S1 and the symbol being read is a one, write a zero, move right and remain in state S1

Turing Machines - Operation

The illustration below depicts a TM which defines a transition from state S1 to S2 when the current symbol being read in a zero.



After the transition has been completed the symbol zero has been replaced with a 1, the read/write head has been moved right and the new state is set to S2



The result of the computation (output) is the sequence of characters left on the tape if and when the Turing Machine halts.

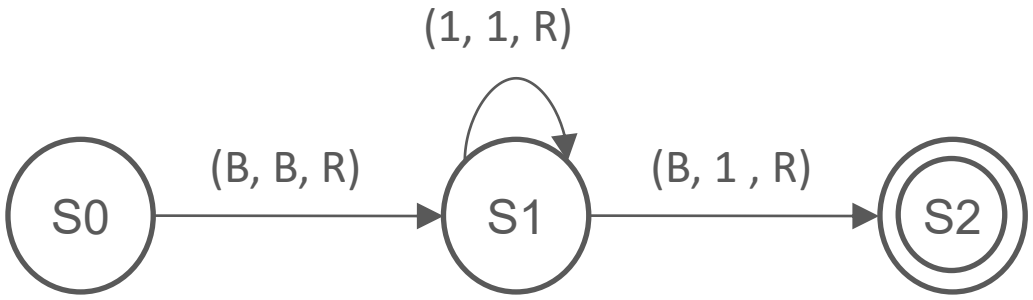
Turing Machines – States

At any given time, a TM is said to be in a particular state. States are usually denoted by the letter S followed by a number e.g. S2 is taken to mean state two.

S0 is conventionally used to denote the initial state. This is the state the TM is in before it starts to operate.

A double circle is used to denote the final or *halting state*. This is the state the TM is in when it finishes.

For example,



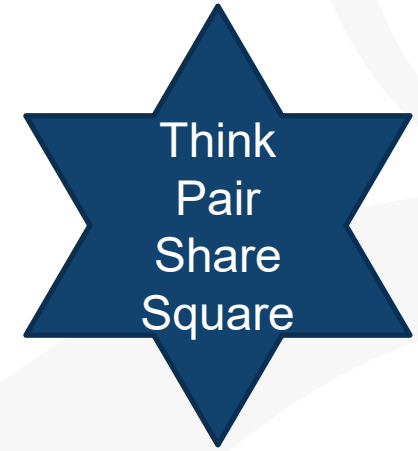
Turing Machines – Significance

Earlier we asked the question:
How do we define computability?

Now we can provide the answer:
A task is computable if it can be carried out by a Turing Machine

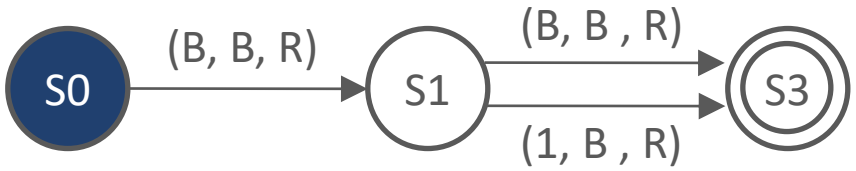
Turing Machine Activity

Each group will trace through the operation of a Turing Machine which will be assigned to them.

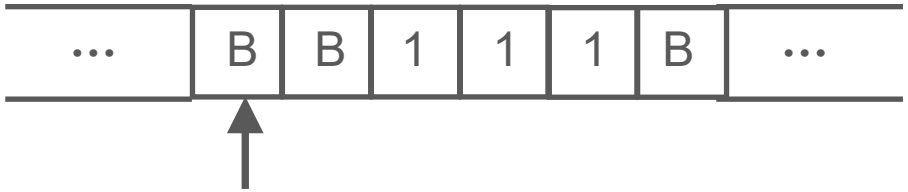


Turing Machines – Activity – Problem #1

Initial State: S0

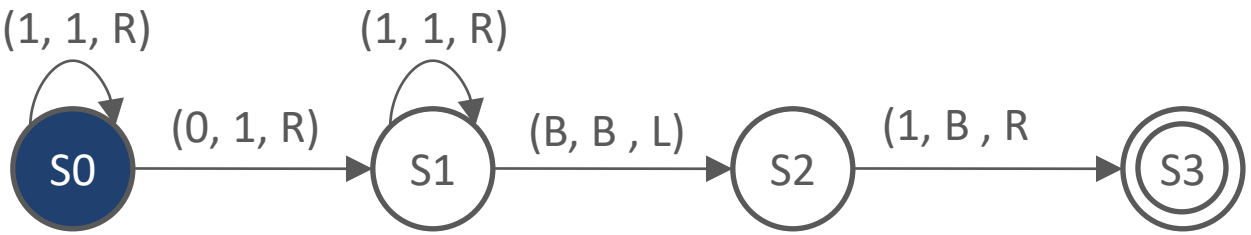


Test input: BB111B

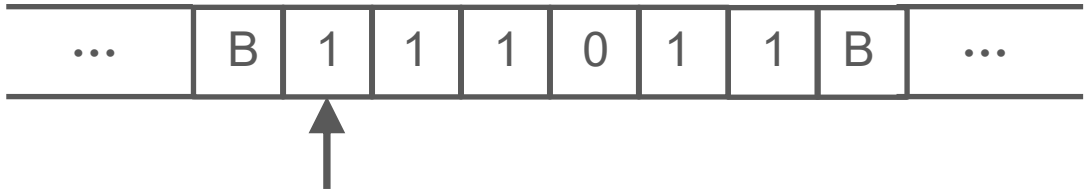


Turing Machines – Activity – Problem #2

Initial State: S0

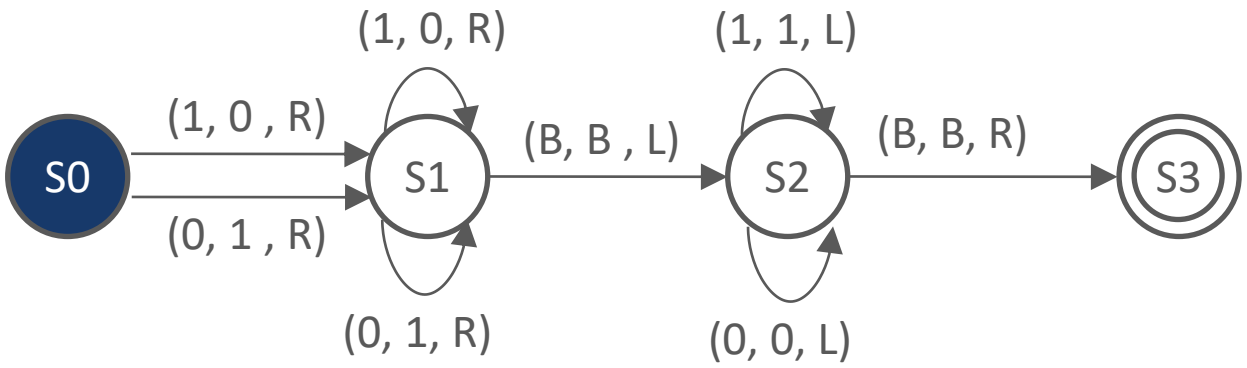


Test input: 111011

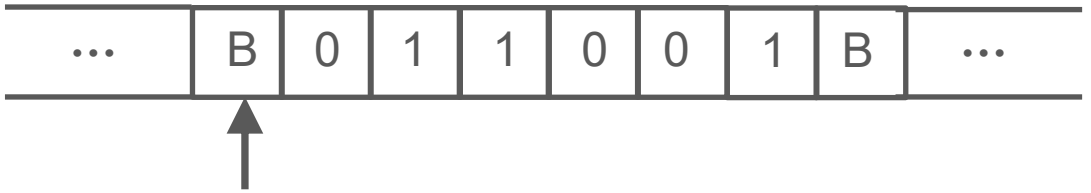


Turing Machines – Activity – Problem #3

Initial State: S0



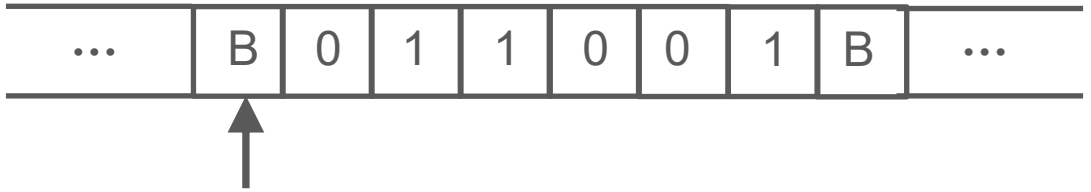
Test input: B011001B



Turing Machines – Activity Handout

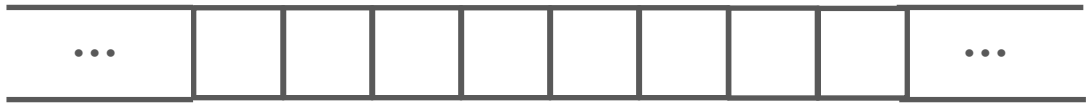
Initial Input

Initial State: S0



After Step 1:

Next State:



After Step 2:

Next State:

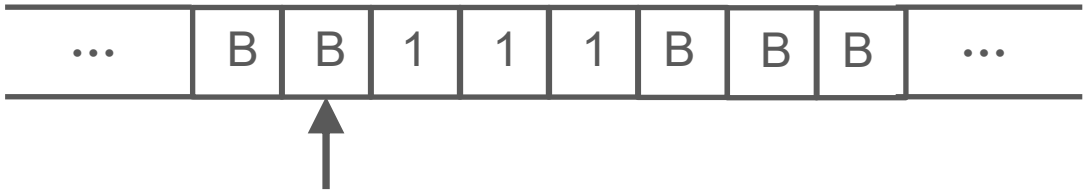
…

…

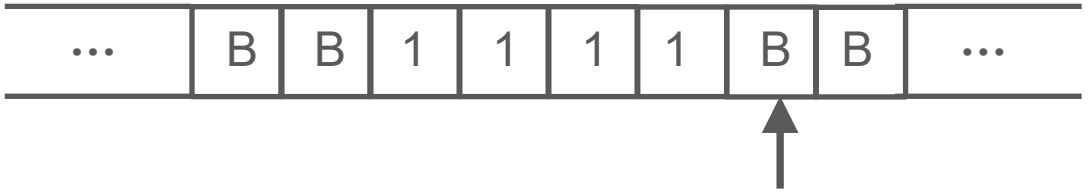
Final Output

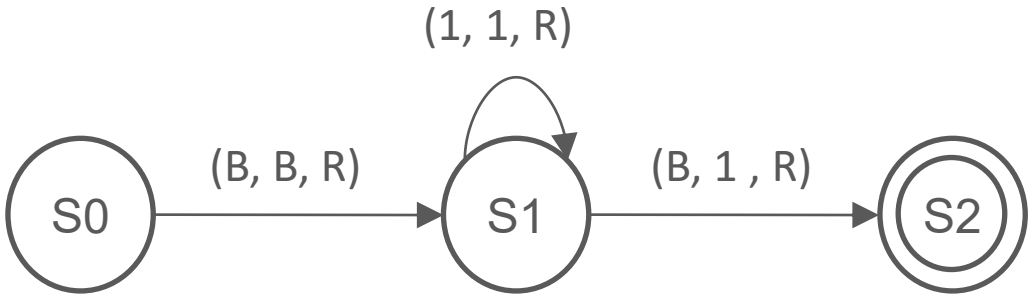
Turing Machines – Example (unary increment)

Test input: 111 (3)



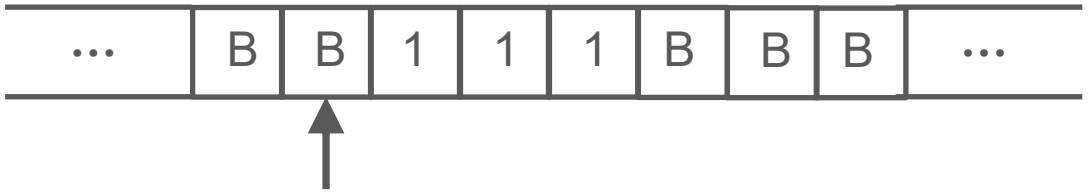
Required output: 1111 (4)



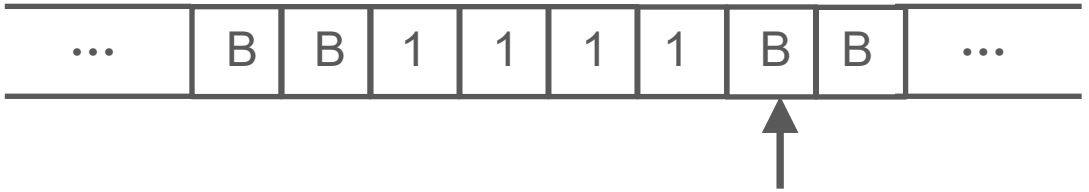


Turing Machines – Example (unary increment)

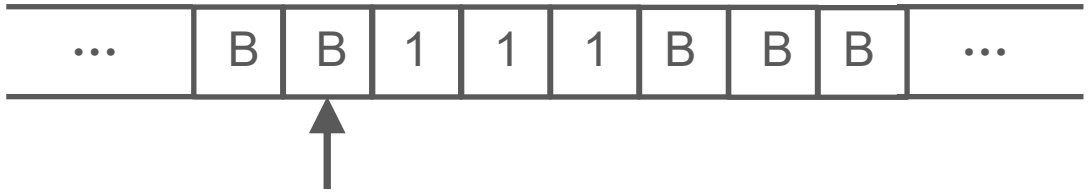
Test input: 111 (3)

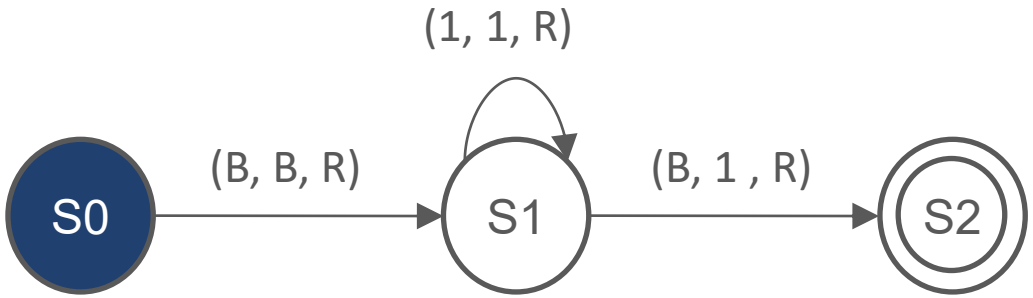


Required output: 1111 (4)



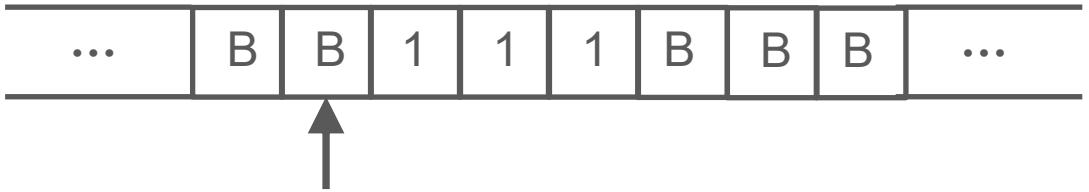
State: S0



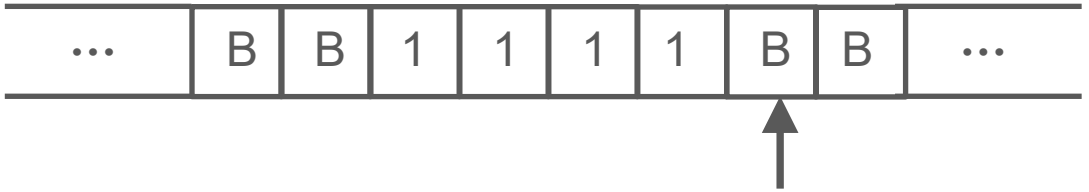


Turing Machines – Example (unary increment)

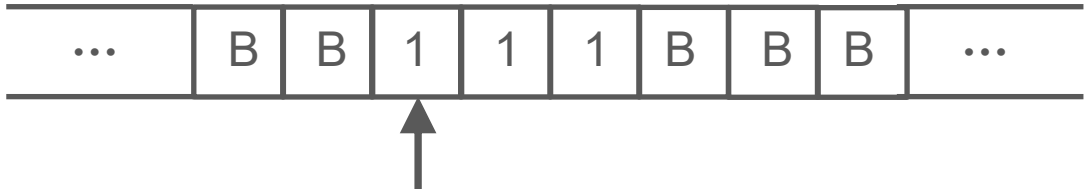
Test input: 111 (3)

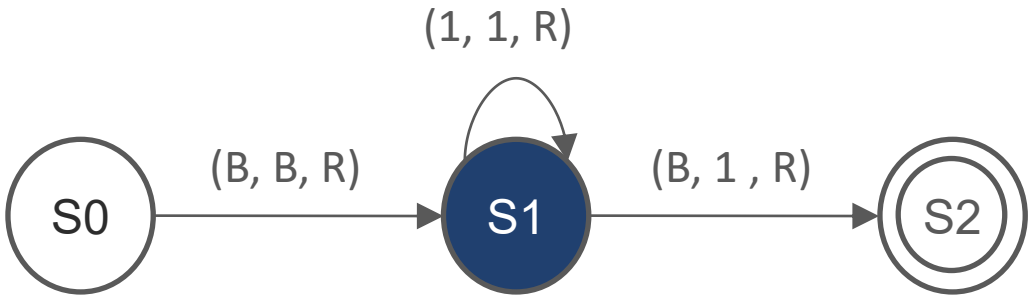


Required output: 1111 (4)



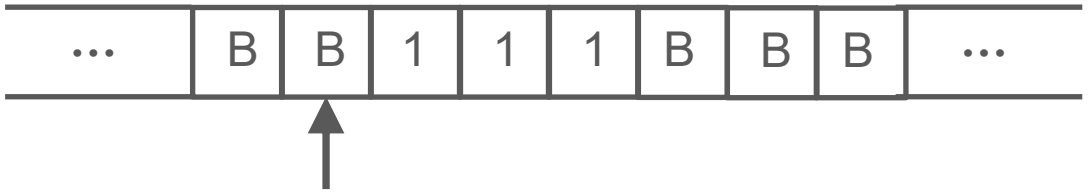
State: S1



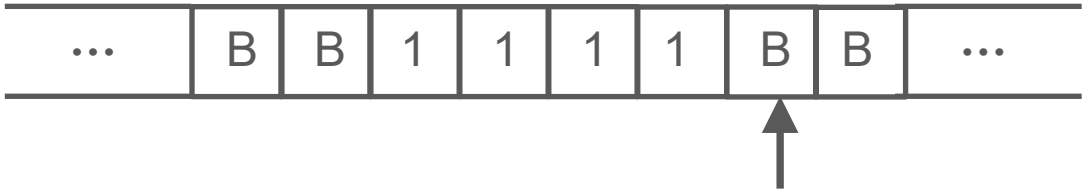


Turing Machines – Example (unary increment)

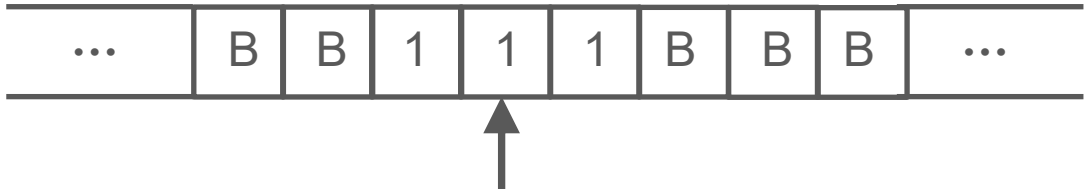
Test input: 111 (3)

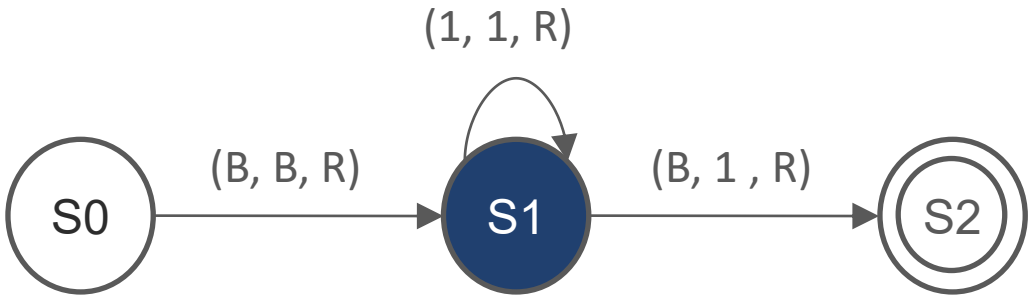


Required output: 1111 (4)



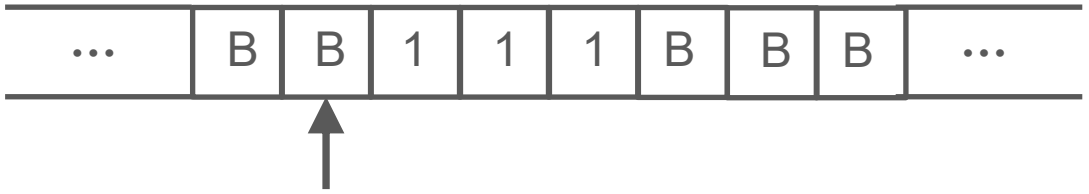
State: S1



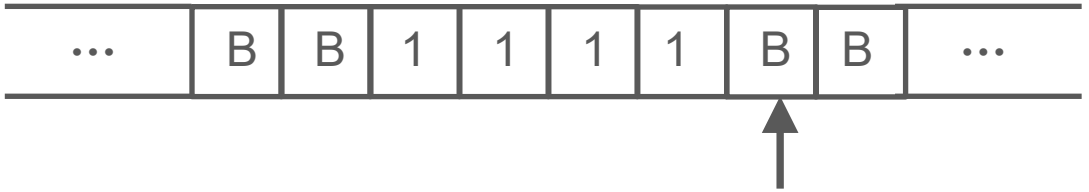


Turing Machines – Example (unary increment)

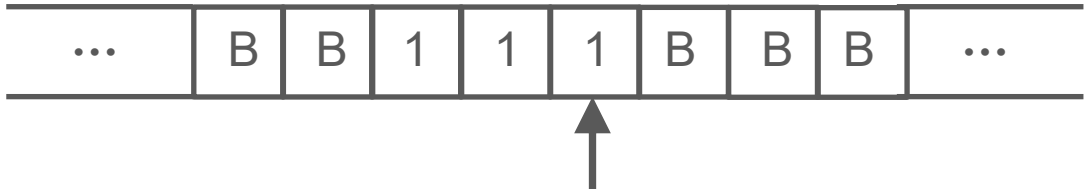
Test input: 111 (3)

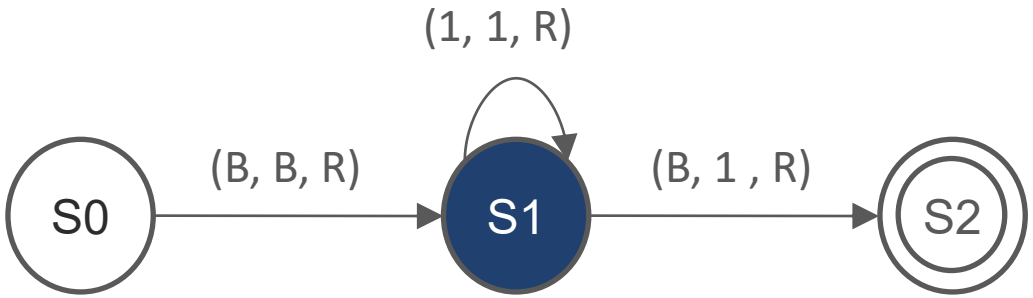


Required output: 1111 (4)



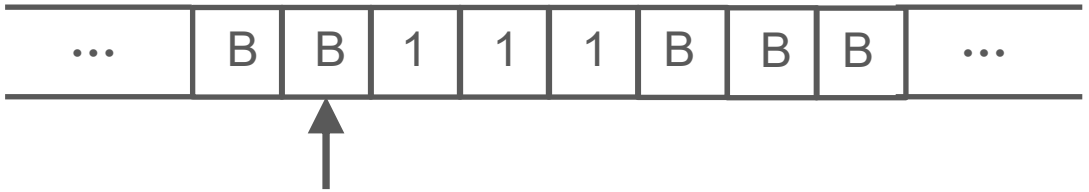
State: S1



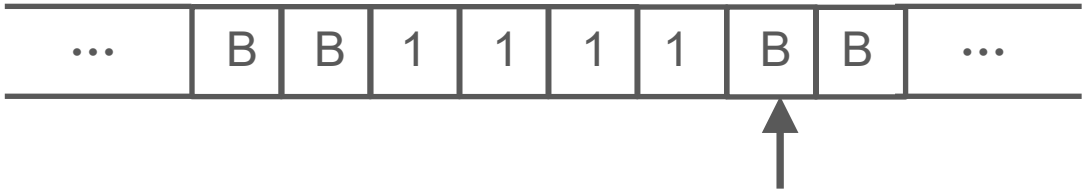


Turing Machines – Example (unary increment)

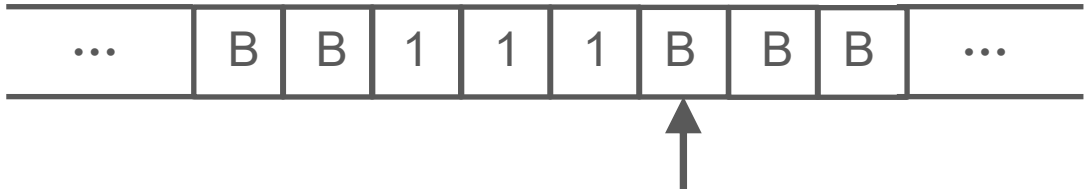
Test input: 111 (3)

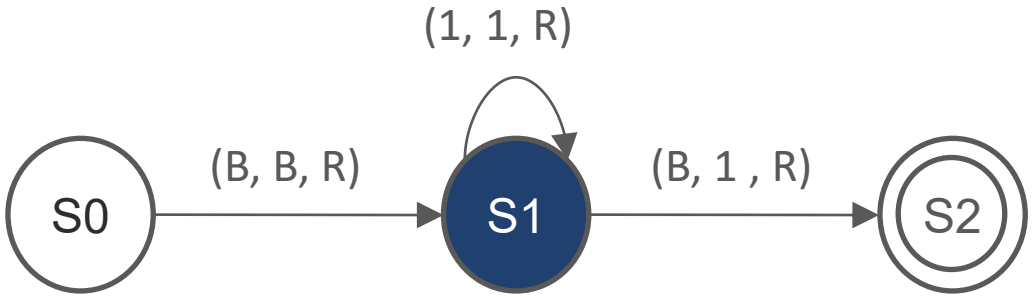


Required output: 1111 (4)



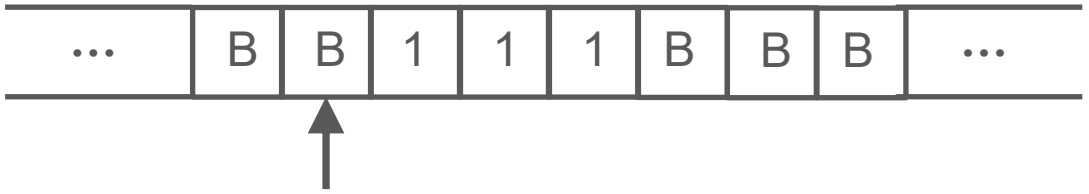
State: S1



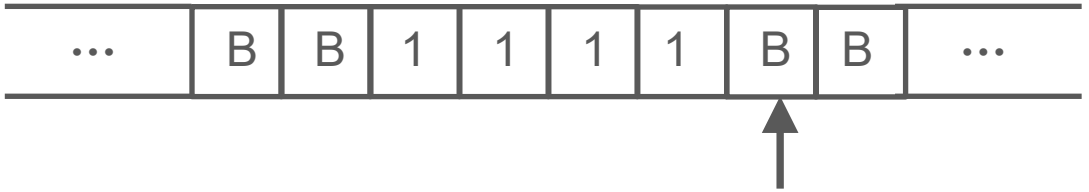


Turing Machines – Example (unary increment)

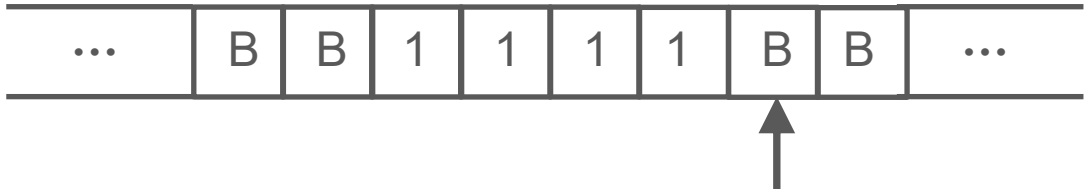
Test input: 111 (3)

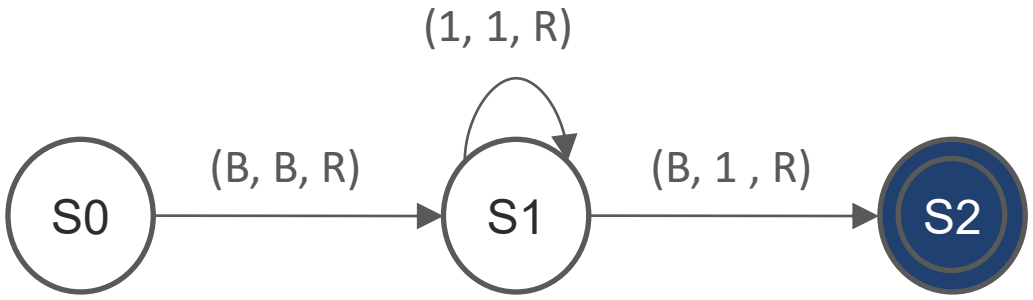


Required output: 1111 (4)



State: S2





PDST
Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

Group Activity

Tick the column that best describes your knowledge in relation to each topic

Topic	Not familiar with topic (not confident to teach others)	Somewhat familiar (novice level understanding)	Very familiar with topic (confident to teach others)	Completely familiar (expert level understanding)
Algorithmic complexity				
Heuristics				
Turing Machines				
Software Development and Management				
Relational Databases/Raw Data				
Recursion				
AI and Machine Learning				
User-centred design				
Communication Protocols				

PDST

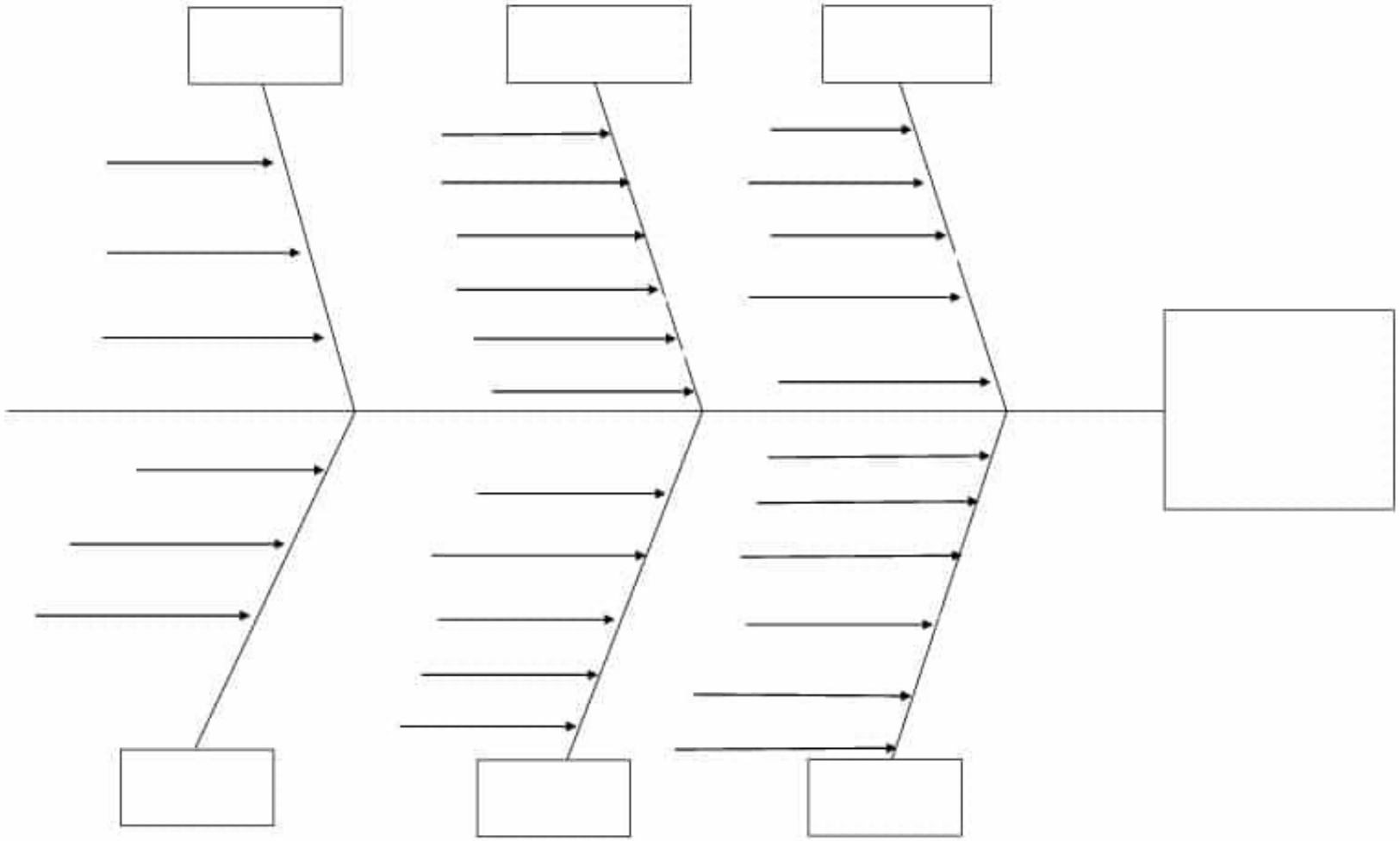
Active Learning

An Integrated Approach to Learning, Teaching and Assessment

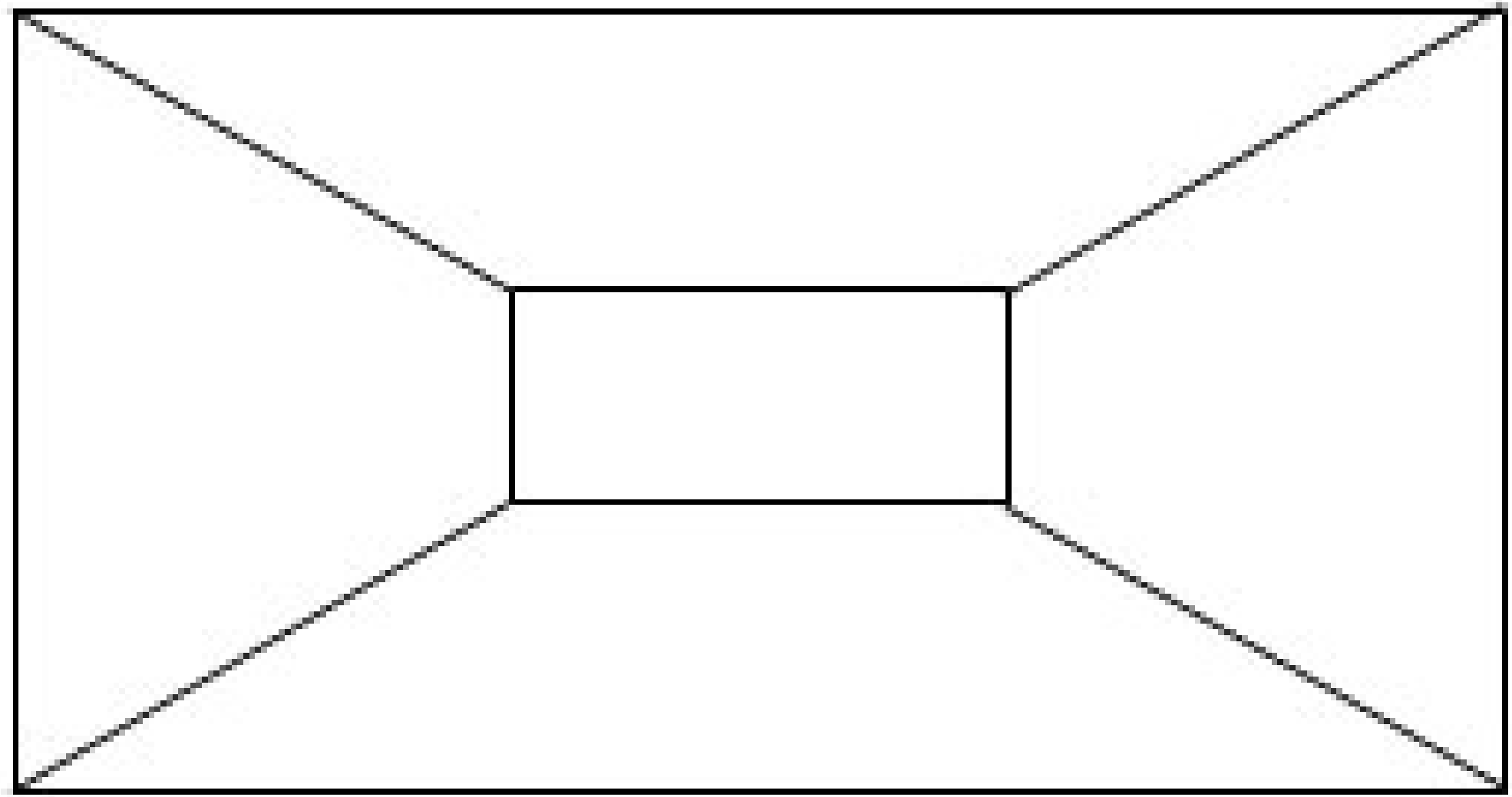
Know?	Want to know?	How will I find out more?	What have I learned?

KWHL – Page 27

Active Learning Tools - Fishbone

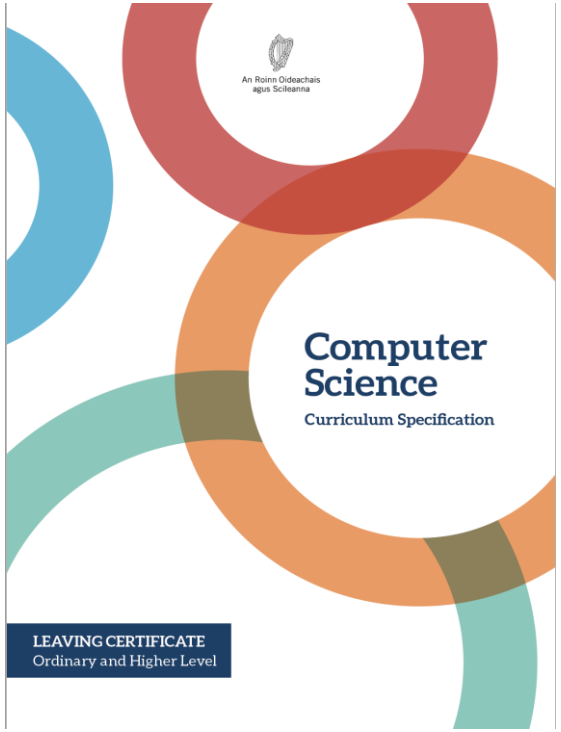


Active Learning Tools - Placemat



Activity: Topics and Groups

- Software Development and Management
- Relational Databases
- Recursion
- User-Centred Design
- AI and Machine Learning
- Communication Protocols



PDST

https:// pdstlccs.padlet.org/cpd/

PDST Professional Development Service for Teachers | An tSeirbhís um Fhorbairt Ghairmiúil do Mhúinteoirí

padlet

Group 1 : 7710g7172nuy

Group 2 : qklzwqf6h9hx

Group 3 : bv1ntbl6arh9

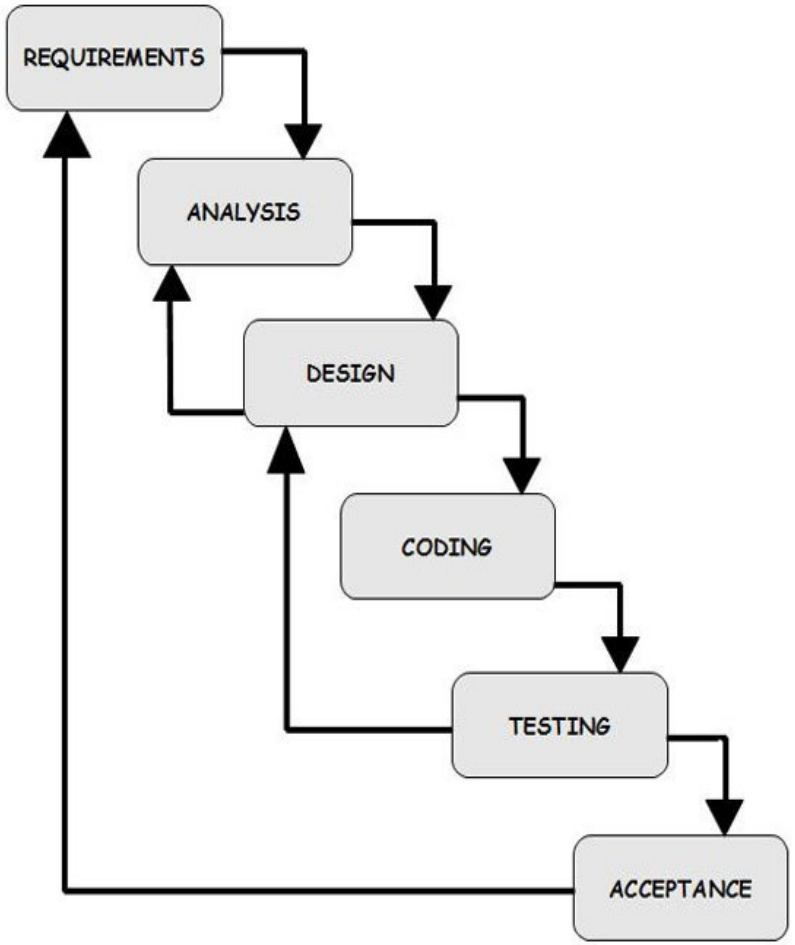
Group 4 : ir6njapfd7k3

Group 5 : o4r1npyac4c6

Group 6 : 7i4lyjo08fee

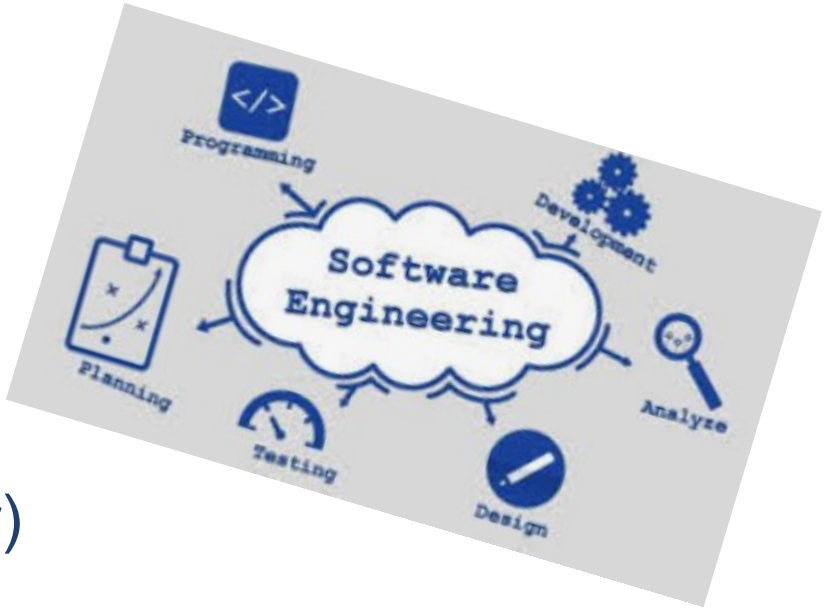
Conclusion

Software Development Methodologies



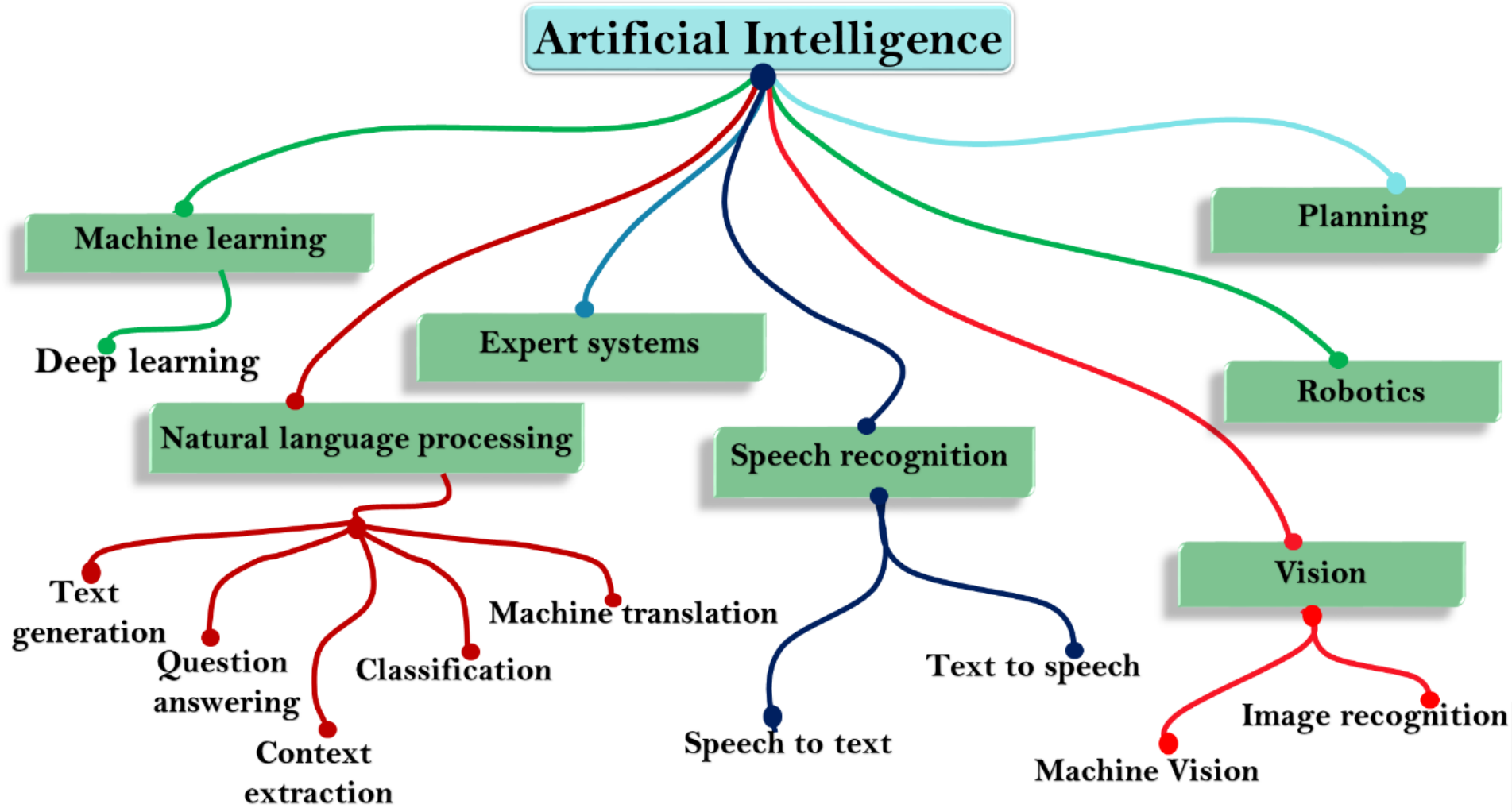
1. Preliminary Analysis – requests are reviewed
   - Deliverable - feasibility analysis document
2. Systems Analysis – if approved, determine the system requirements for new system
   - Deliverable – systems requirement document
3. Systems Design – converts system analysis requirements into system design document deliverable
4. Programming – coding commences using design documents
5. Testing – ensures that the code functions according to requirements
6. Implementation – converting from old system to new system
   - Training, documenting functions, and data conversion
7. Maintenance – support for reporting prioritizing, and fixing bugs

Roles and Resposnibilities



- Project Sponsor
- Project Manager
- Analyst (Business Analyst)
- Designer (Graphic Designer, UX Designer)
- System Architect (Technical Lead)
- Programmer (Developer, Software Engineer)
- Tester (QA Engineer)
- Technical Writer

Links between Artifical Intelligence and Machine Learning



Source: https://www.javatpoint.com/subsets-of-ai

Relational Databases

Order #	Item #	Item Name	Customer ID	Customer Name	Customer Address	Item Description
1	3	Plates	2	Tom	123 Bear Dr.	decorative plates
2	2	Cups	3	Diane	212 Bear Ave.	decorative cups
3	3	Plates	3	Diane	212 Bear Ave.	decorative plates
4	1	Silverware	1	Meg	321 Bear Cr.	decorative silverware
5	4	Napkins	4	Dylan	543 Bear St.	decorative napkins
6	3	Plates	1	Meg	321 Bear Cr.	decorative plates

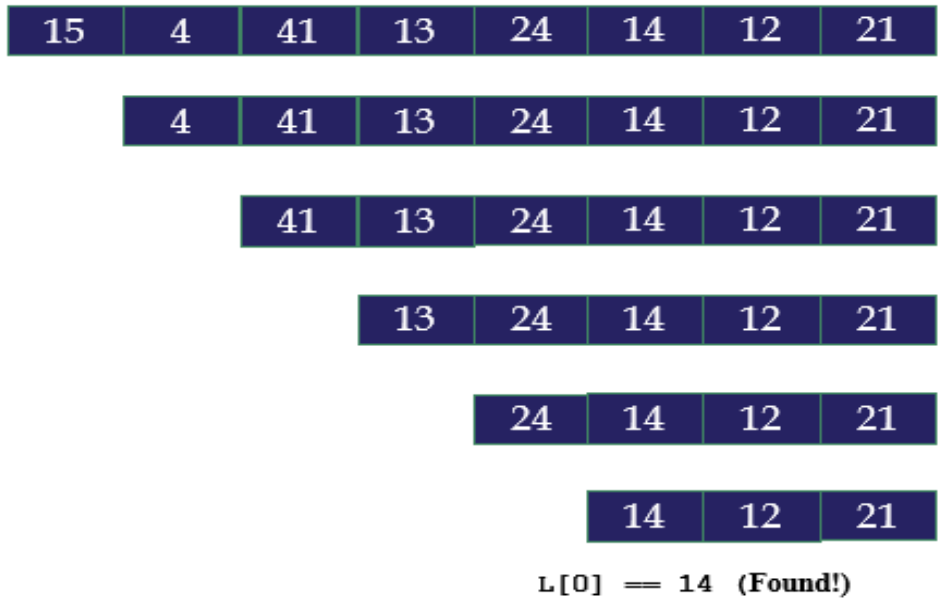
- **Database:** Any collection of data
- **Relational Database:** A structured collection of related data stored in tables
- **Table:** A set of data elements (values) organised by rows (records) and columns (values)
- **Attribute:** A characteristic of the data in the table, describing a field or cell in a table.
- **Primary Key:** A unique identifier for a row in a table
- **Foreign Key:** An attribute in a table that is used as a primary key in another table i.e. provides the relationship by linking one table to another

Recursion

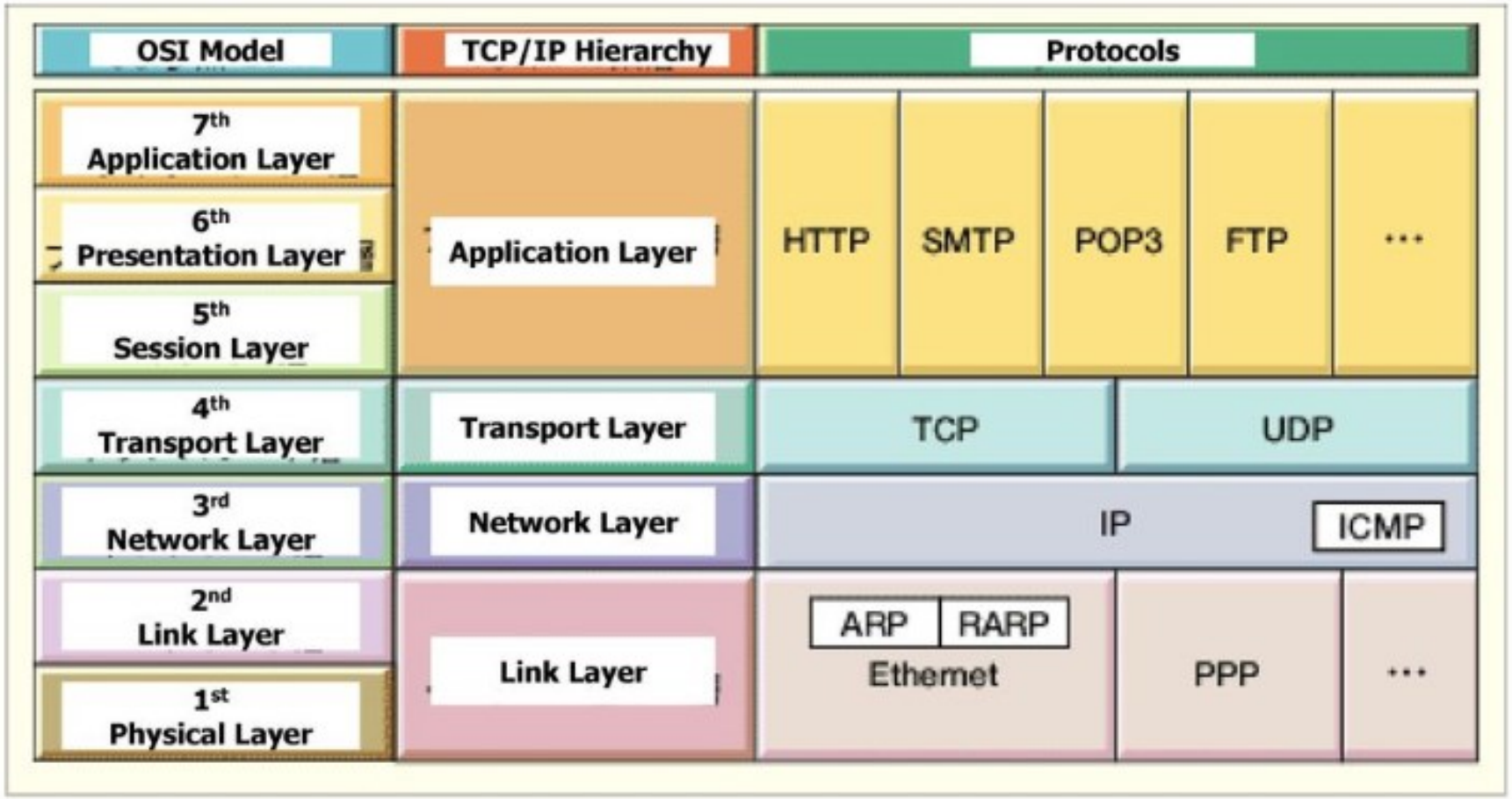
```
def linear_search_v6(v, L, index=0) :
    if len(L) != 0:
        if L[0] == v:
            return index

        r =  linear_search_v6(v, L[1:], index+1)
        if r != -1:
            return r

    return -1
```



Communication Protocols



Link Layer : includes device driver and network interface card
Network Layer : handles the movement of packets, i.e. Routing
Transport Layer : provides a reliable flow of data between two hosts
Application Layer : handles the details of the particular application

Source: https://slideplayer.com/slide/13069007/

An Roinn Oideachais agus Scileanna
Department of Education and Skills