4COSC002W Mathematics for Computing

Lecture 3

Introductions to Algorithms. Collecting and Representing Data. Functions.

UNIVERSITY OF WESTMINSTER

Revision: Cartesian Product

The **Cartesian product** of two sets is the set of all ordered pairs where the first element in each pair is from the first set, and the second element is from the second set. The Cartesian product of sets A and B is denoted as **A×B**.

Example:

A={1,2} and B={a,b}; A×B={(1,a),(1,b),(2,a),(2,b)}

Order Matters!

The order of sets in a Cartesian product is significant. A×B is not necessarily the same as B×A.

For instance, using the sets from the above example,

B×A={(a,1),(a,2),(b,1),(b,2)}, which is different from A×B.

Introduction To Algorithms

Algorithms. Flowcharts. Math Interpretation of Search and Sorting Algorithms

Algorithms

Definition: A finite sequence of well-defined instructions for completing a specific task.

Application: Used in computer programming for *data processing*, *calculation*, and other related operations.

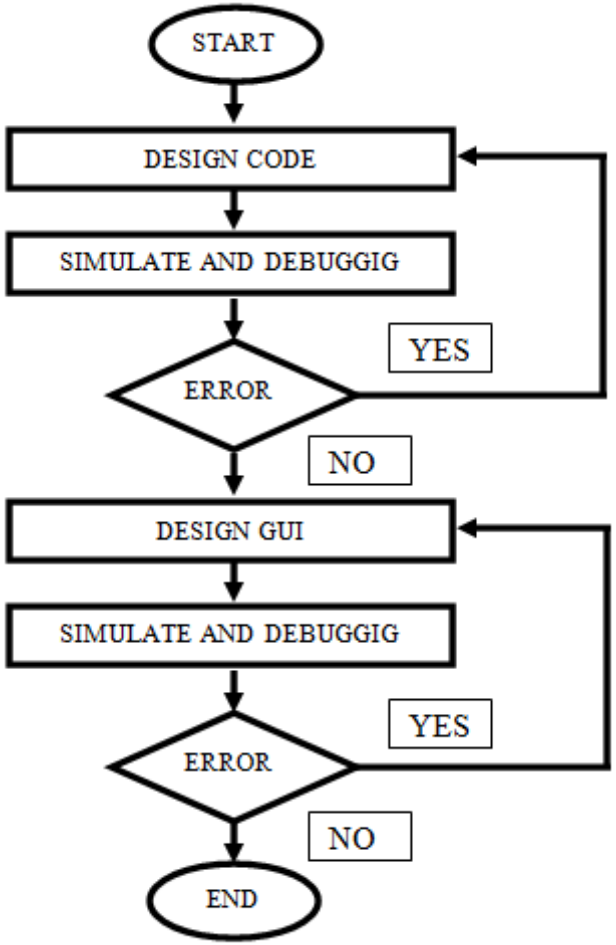
Why Algorithms?

- **Efficiency**: Ensures tasks are done more efficiently and faster.
- **Problem-solving**: Provides a *systematic way* to solve problems and achieve desired outcomes.
- **Resource Optimization**: Manages and utilises computational resources in an optimal way.

Types of Algorithms:

1. **Searching Algorithms**: Aimed at efficiently retrieving information from a data structure.
2. **Sorting Algorithms**: Intended to rearrange data in a particular order efficiently

Algorithm Visualization: Flowcharts



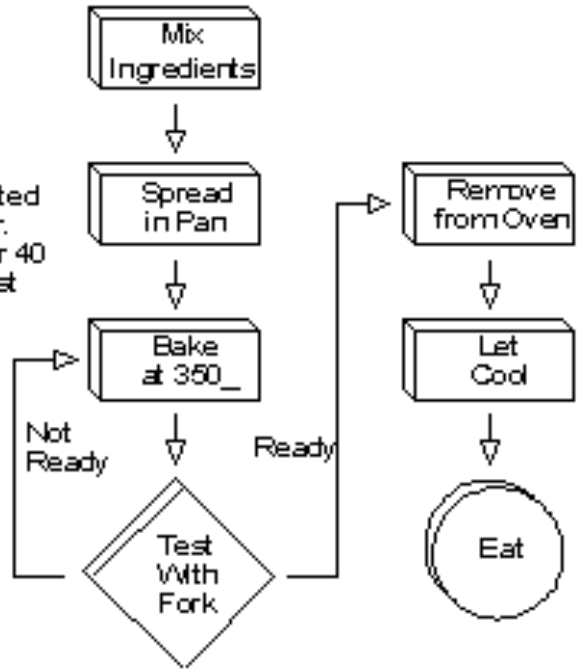
Recipe
CHOCOLATE CAKE

4 oz. chocolate	3 eggs
1 cup butter	1 tsp. vanilla
2 cups sugar	1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

```
Declare variables:
    chocolate   eggs        mix
    butter      vanilla
    sugar       flour
mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
bake (mix, 350)
```



Types of Algorithms

1.Searching Algorithms

Purpose: Quickly identify and locate an item in a dataset.

Key Examples:

1. **Linear Search**: A straightforward method that checks each element.
2. **Binary Search**: A fast method that works by repeatedly dividing the portion of the array in half that could contain the target item.

2.Sorting Algorithms

Purpose: Arrange data in a specified order to streamline further operations.

Key Examples:

1. **Bubble Sort**: Simple but potentially slow, works by repeatedly stepping through the list, comparing each pair of adjacent items and swapping them if they are in the wrong order.
2. **Quick Sort**: Efficient works by dividing a large array into two smaller arrays, the low elements and the high elements, and recursively sorts them.

Linear Search

Given a set S={a_1, a_2 ,…, a_n} and a target value *d*.
For i=1 to *n*, if a_i=d, then return i. If no such i is found, return "Not Found."

Example:

Given a list of 12, 34, 25, 6, 17. Let's search for the number 25.

Step	Current Element	Is it equal to 25?
1	12	No
2	34	No
3	25	**Yes**

So, the linear search finds 25 in 3 steps.

Binary Search

Given a *sorted* set S={a_1, a_2, ..., a_n} and a target value d. Let L=1 and R=n. While L≤R,

1. Let $M = \lfloor \frac{L+R}{2} \rfloor$
2. Let If a_M=d, return M.
3. Else if a_M<d, set L=M+1
4. Else, set R=M−1.

If there is no solution, return "Not Found."

Example: Binary Search

Suppose we have the following *sorted* array: 3,7,11,23,27,31,37,43,47, and we want to find the element 37.

Step	Low	High	Middle	Middle Element	Comparison	Found?
1	1	9	5	27	27 < 37	No
2	6	9	7	37	37 = 37	Yes

Step 1: Low=1, **High=9** (since there are 9 elements in the array). **Middle** is calculated as: [(1+9)/2]=5

Middle Element=27 (element at the 5th position in the array). 27 < 37, so we will search in the right sub-array, meaning we update Low=Middle+1 (i.e., Low=6).

Step 2: Low=6, **High=9**. **Middle** is calculated as: [(6+9)/2]=7

Middle Element=37 (element at the 7th position in the array). 37 = 37, so we have found the element.

Example: Bubble Sort

Given a set S={a_1, a_2, …, a_n}. For i=1 to n, For j=1 to n−i, If $a_j > a_{j+1}$, swap (a_j,aj+1).

Example: Given a list: 29, 10, 14, 37, 13. Let's sort it using bubble sort.

Step	List Before	Element 1	Element 2	Swap Needed?	List After
1	29,10,14,37,13	29	10	Yes	10,29,14,37,13
2	10,29,14,37,13	29	14	Yes	10,14,29,37,13
3	10,14,29,37,13	29	37	No	10,14,29,37,13
4	10,14,29,37,13	37	13	Yes	10,14,29,13,37
5	10,14,29,13,37	10	14	No	10,14,29,13,37
..	..	..	..	..	..

Collecting and Representing Data in Mathematics

Category	Type	Description
1. Tabular	Frequency Tables	Organize data to show the frequency of different outcomes.
	Contingency Tables	Display relationships between two+ categorical variables.
2. Graphical	Bar Graphs, Histograms	Visualize categorical and frequency data with bars.
	Scatter Plots, Line Graphs	Explore relationships and trends in numerical data.
	Pie Charts	Depict parts-to-whole relationships in data.
3. Algebraic	Equations	Define relationships between variables algebraically.
	Inequalities	Show the relative positions of numerical values.
	Functions	Describe dependent relationships between variables.

Relations and Functions

Relations vs Functions. Basic Functions Types. Recursive Functions. Inverse of a function

Relations

A **relation (R)** from set A to set B is defined as a *subset of the Cartesian product* $A \times B$. In simpler terms, a relation R consists of ordered pairs (a,b) such that a is from set A and b is from set B. Thus, a Relation can be represented as $R \subseteq A \times B$.

Example: If $A=\{1,2\}$ and $B=\{x,y\}$, then $A \times B=\{(1,x),(1,y),(2,x),(2,y)\}$

Sets involve well-defined collections of objects, and relations extend this to how elements from different sets interact or are associated.

Computational Realms: can represent networks or sets of data points in databases, social networks, or any system where elements might be interconnected or paired in a *non-unique* manner.

Ways to Represent a Relation

Ordered Pairs

$(2, -1)$

$(1, -2)$

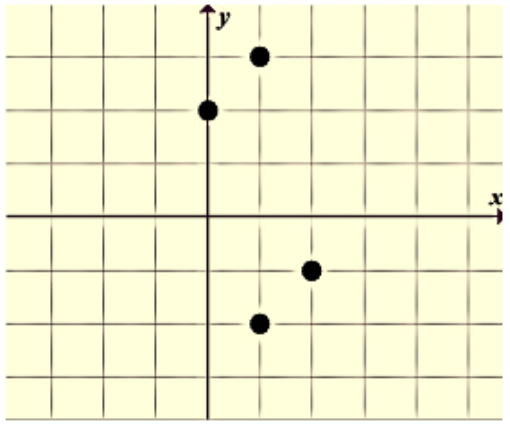
$(1,3)$

$(0,2)$

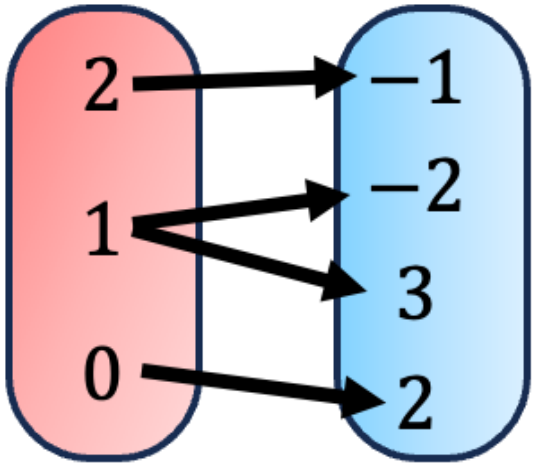
Table

x	y
2	-1
1	-2
1	3
0	2

Graph



Mapping Diagram



Functions

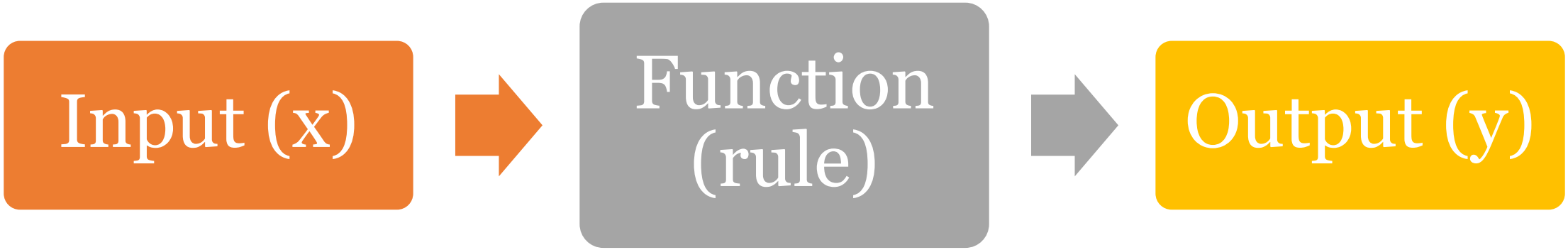
A **function (f)** from set A to set B is a specific type of relation with an additional constraint: each element from set A must be associated with *exactly one element* in set B. In other words, *function (f)* from set A to set B is a rule that assigns to each element $a \in A$ exactly one element $b \in B$, denoted $f: A \rightarrow B$.

Is Function a Relation? Yes, every function is a relation, but not every relation is a function.

Example: a function might be f={(1,x),(2,y)}, but not f={(1,x),(1,y),(2,y)} because '1' from set A is associated with two elements in set B.

Computational Realms: utilised in *algorithm creation*, procedural mappings, and data processing in computational settings where unique input-output mappings are vital.

Visual Representation of a Function



x=1
x=14

x+3

y=1+3=4
y=14+3=17

Domain, Rule, Co-Domain

Domain: The **domain** of a function is the set of *all possible input values* (or "arguments") that the function is defined to accept.

In Context: In algorithms or computational processes, understanding the domain is crucial for determining the range of acceptable inputs and preventing errors during computation.

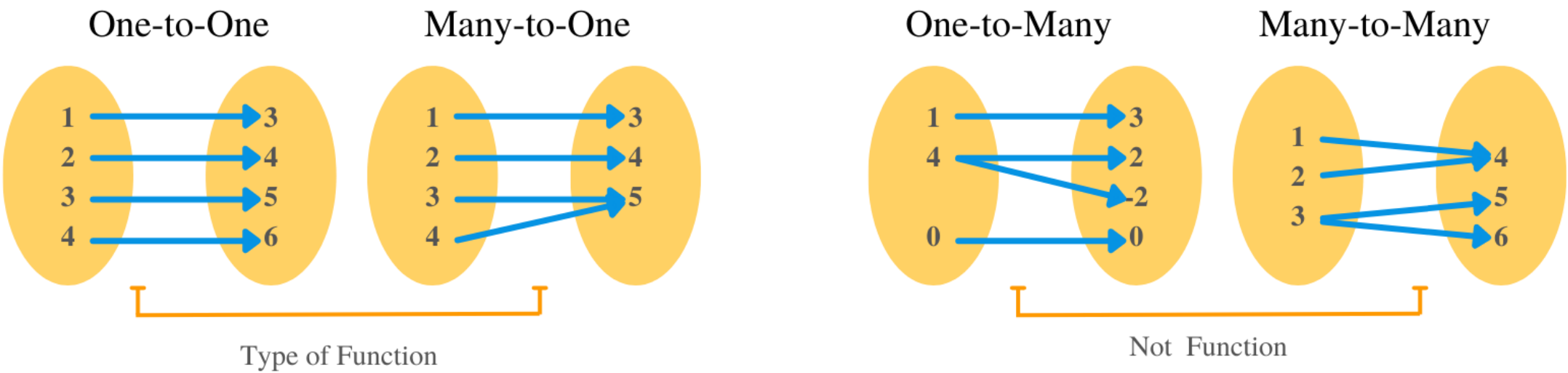
Co-domain: The **co-domain** of a function is the set of *all possible output values*. Not to be confused with the "*range*," the co-domain includes all potential outputs, whether they are achievable with the given function rule or not.

In Context: In computing, defining the co-domain helps to understand the scope of possible outputs and can aid in resource allocation and managing memory usage effectively.

Rule: The **rule** of a function is the specific manner in which inputs (from the domain) are associated with outputs (in the co-domain).

In Context: In computational algorithms, the rule is the procedure or formula applied to the input data to generate outputs. Ensuring this rule is well-defined is essential for algorithm effectiveness and predictability.

Functions vs Relations



Inverse of a Function

In mathematics, the **inverse** of a function, denoted as f^{-1}, is a function that "reverses" the input-output mapping established by the original function f.

In formal terms, a function $f: A \to B$ has an inverse $f^{-1}: B \to A$ if and only if the following conditions hold for all $x \in A$ and $y \in B$: $f(f^{-1}(y)) = y$ and $f^{-1}(f(x)) = x$

Key Points:

- **Bijectiveness:** To have an inverse, a function must be **bijective**, meaning it is both injective (one-to-one) and surjective (onto).
  - **One-to-One (Injective):** No two different elements in A are mapped to the same element in B.
  - **Onto (Surjective):** Every element of B is the image of at least one element in A.

Example: Inverse

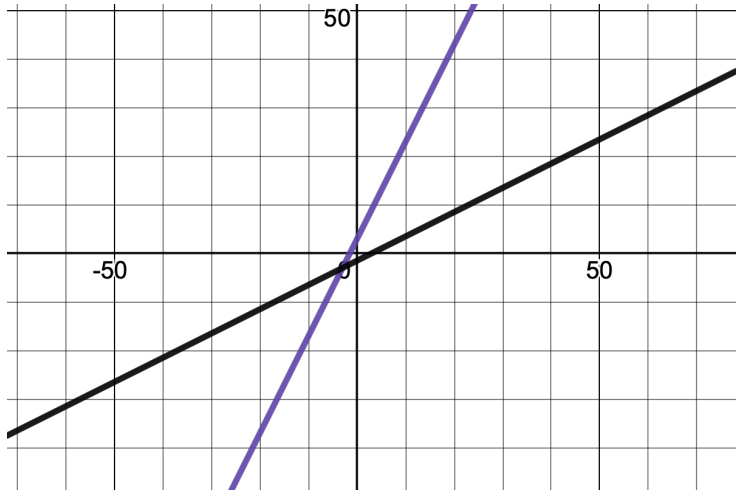
Let's take a simple example: **f(x)=2x+3**, with a domain and codomain in the set of all real numbers (R).

To find the inverse, we swap the roles of y and x and solve for y:
1) y= 2x+3; **2)** swap x and y: x=2y+3; **3)** solve for y: $y=(x-3)/2$

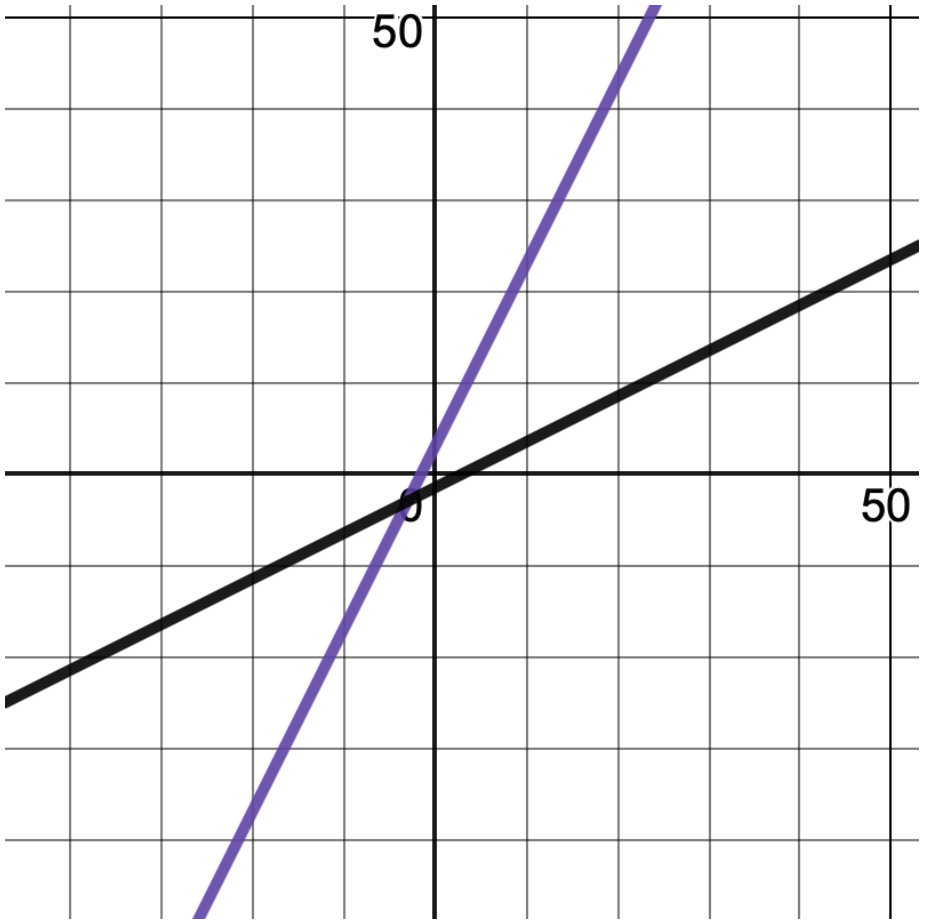
Hence, the inverse function $\mathbf{f^{-1}(x)=(x-3)/2}$.

Checks:

$$f(f^{-1}(x)) = f\left(\frac{x-3}{2}\right) = 2\left(\frac{x-3}{2}\right) + 3 = x$$
$$f^{-1}(f(x)) = f^{-1}(2x+3) = \frac{2x+3-3}{2} = x$$



Types of Functions: Linear Function



Definition (1): **f(x)=mx+b**, where **m** is the slope and **b** is the y-intercept.

Graph: A straight line.

Growth: Constant rate.

Inverse: $f^{-1}(x)=(x-b)/m$

Applications: Simple rate problems, budget calculations, etc.

Slope of a Line

The **slope (m)** of a line describes its steepness, incline, or grade. Mathematically, it represents the *rate of change of a functio*n.

Formula: $m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$ where Δy and Δx represent changes in the y and x directions, respectively.

- **Positive Slope:** Line ascends as x increases.
- **Negative Slope:** Line descends as x increases.
- **Zero Slope:** Horizontal line.
- **Undefined Slope:** Vertical line.

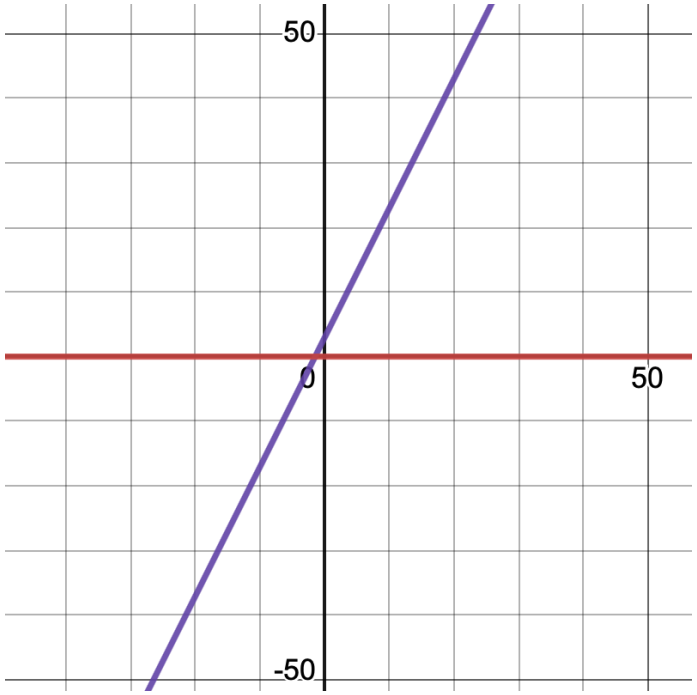
Applications: Determining the rate of change and trend in data sets. Understanding direction and steepness in graphical representations of functions.

Intersecting with the X-Axis

When a line intersects the x-axis, the y-coordinate is zero. Therefore, to find the x-intercept, we set y=f(x)=0 and solve for x.

0=mx+b; solving for *x*, we get:

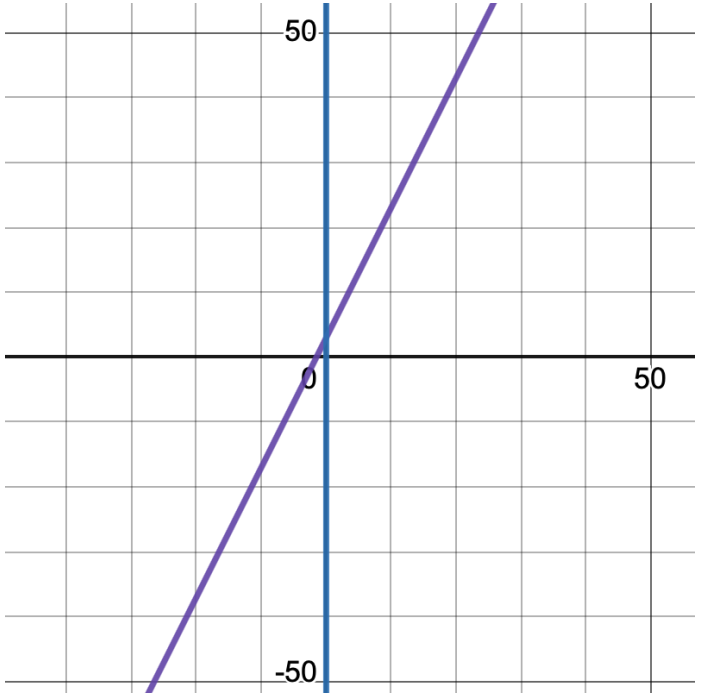
x=–*b*/m; so, the x-intercept is (–b/m,0).



Intersecting with the Y-Axis

The y-intercept is found when x=0. By substituting x=0 into equation (1), we get:

$f(0)=m\cdot 0+b=b$: so, the y-intercept is (0,b).



Intersection of Two Functions

The **intersection points** of two functions, f(x) and g(x), occur where f(x)=g(x).

Mathematical Solution:

1. Set the two functions equal to each other: f(x)=g(x).
2. Solve for x to find the x-coordinates of the intersection point(s).
3. Plug x back into either function to find the corresponding y-coordinate(s).

Applications:

- Finding common solutions to different mathematical models.
- Analysing where two data sets or trends coincide.

Example: Intersection of Two Functions

Let's investigate the intersection points of the following simple functions:

$f(x)=x+2$ **(1)**

$g(x)=x^2$ **(2)**

To identify the intersection points, we equate f(x) and g(x):

$x+2=x^2$

Example: Intersection of Two Functions

Solve for x
Rearranging the equation, we aim to find the solutions for x:
$x^2 - x - 2 = 0$

Factoring the equation gives us the following:
$(x-2)(x+1)=0$

This yields two solutions for x:
1. $x=2$
2. $x=-1$

Example: Intersection of Two Functions

We substitute these x-values into the original equation (1) or (2) to identify the corresponding y-values.

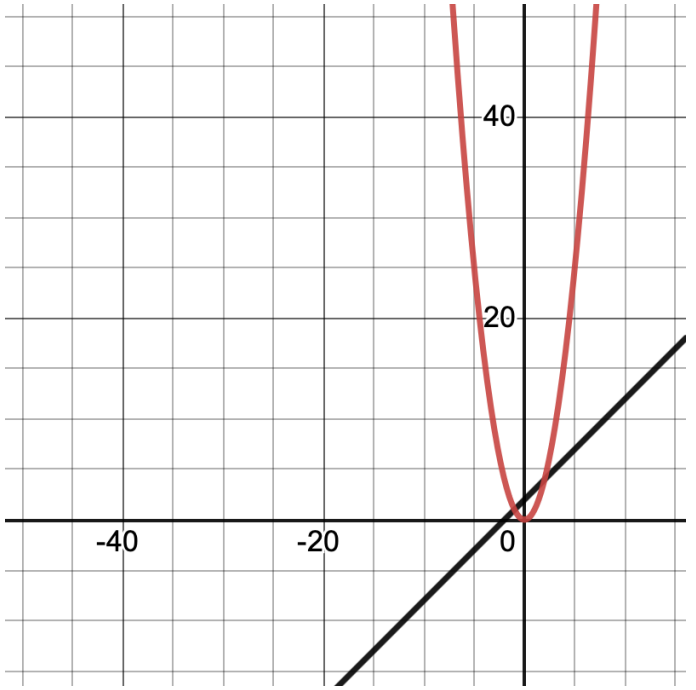
Using equation (1):

- When x=2: $f(2)=2+2=4$

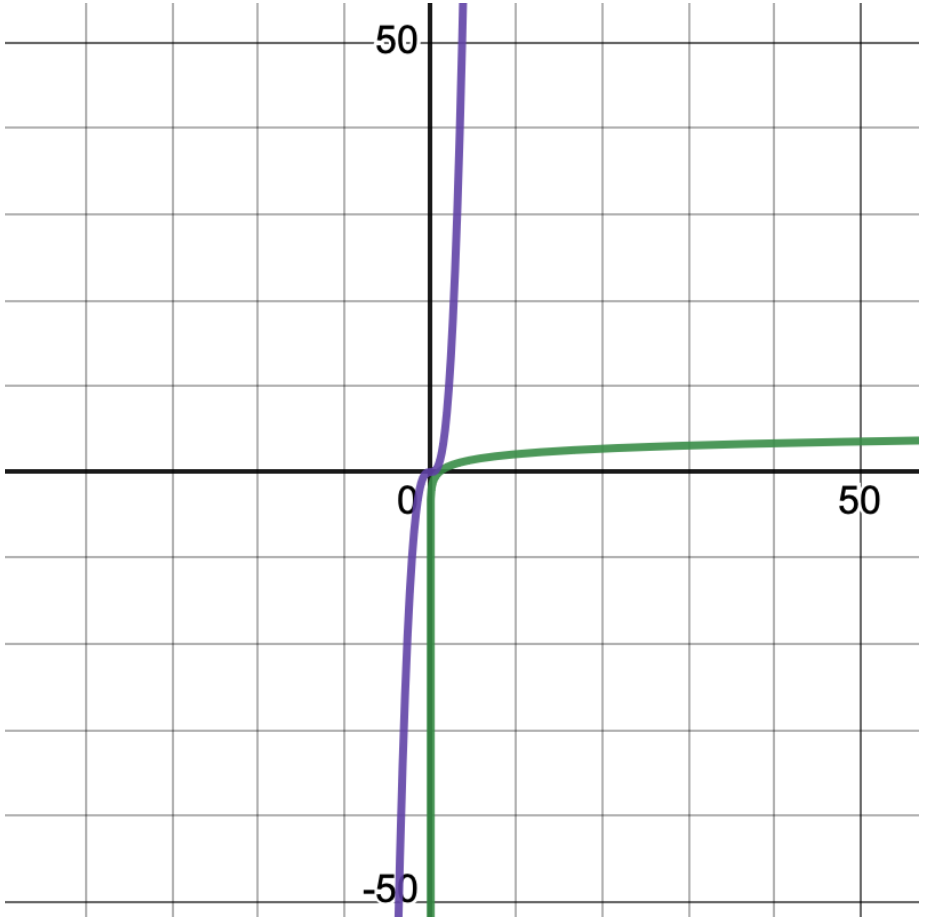
Intersection point: (2, 4)

- When x=−1: $f(-1)=-1+2=1$

Intersection point: (-1, 1)



Types of Functions: Logarithmic Function



Definition: $f(x)=\log_b(x)$ where b is the base.

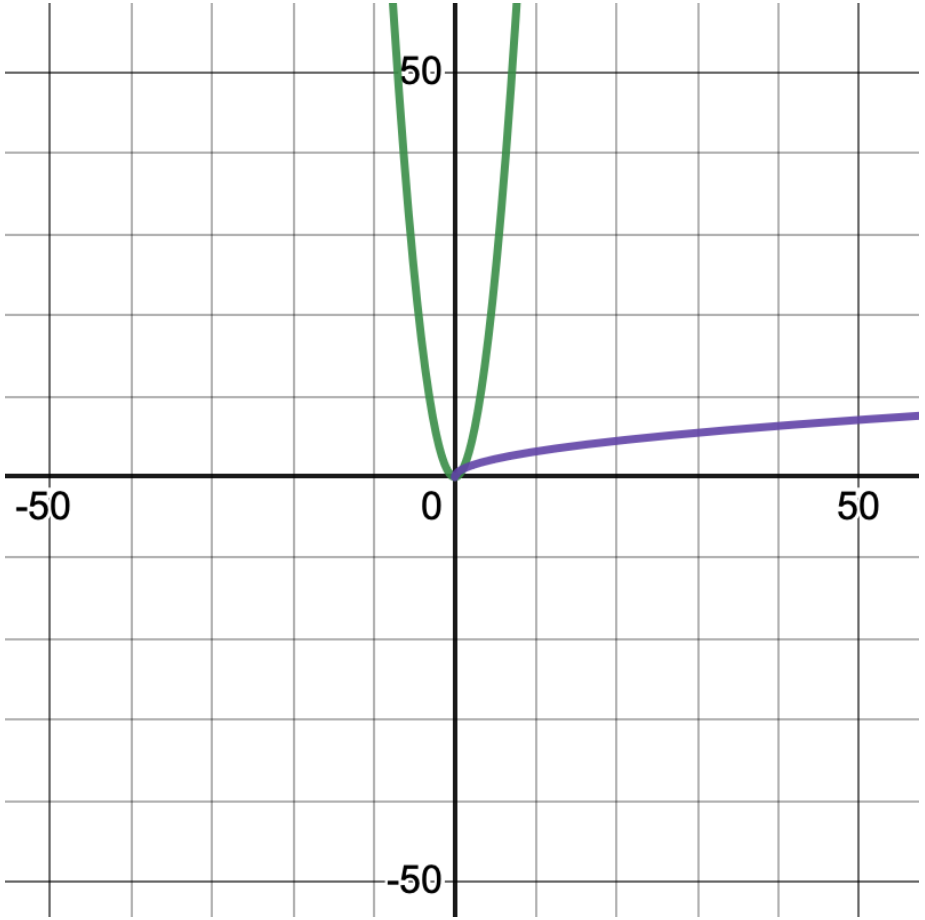
Graph: An increasing curve, asymptotic to the x-axis.

Growth: Slow and increases as x increases.

Inverse: $f^{-1}(x)=b^x$

Applications: Solve for exponents, information theory, etc.

Types of Functions: Squared Function



Definition: $f(x)=x^2$

Graph: A parabola opening upwards, vertex at the origin (0,0).

Symmetry: Symmetrical about the y-axis

Growth: As |x| increases, f(x) increases rapidly.

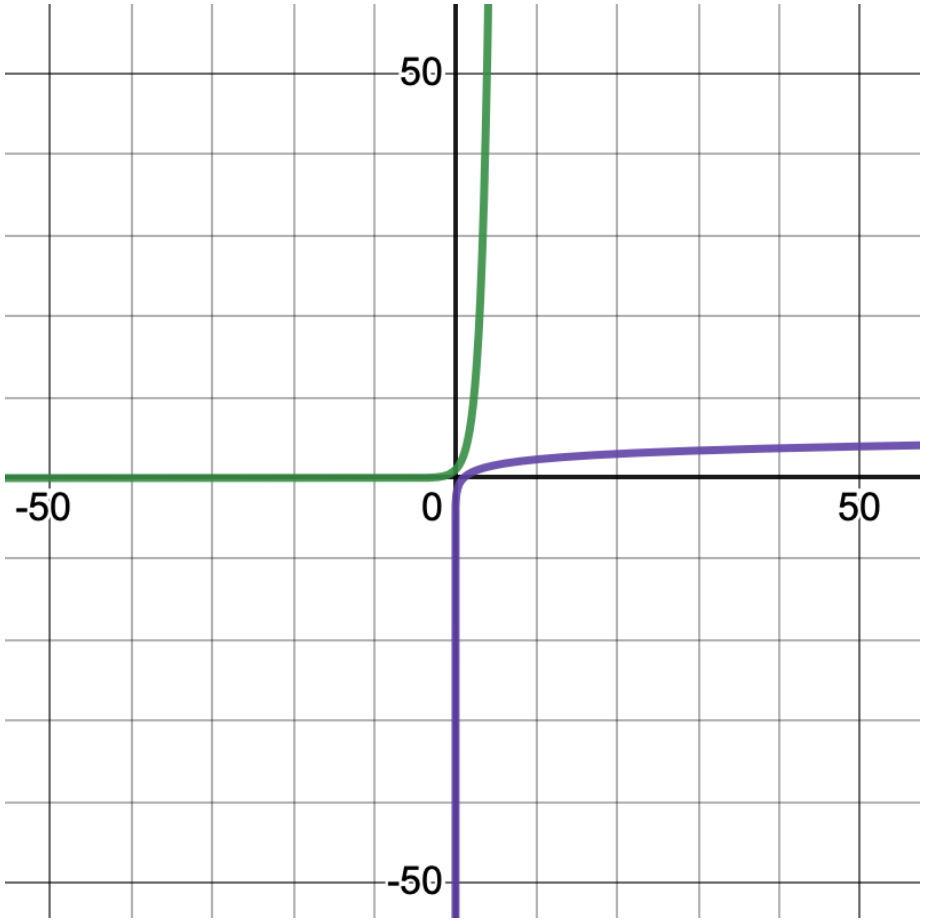
Inverse*:

$f^{-1}(x)=\sqrt{x}$ (when domain of f is $x\geq 0$)

$f^{-1}(x)=-\sqrt{x}$ (when domain of f is $x\leq 0$)

*The squaring function does not have an inverse function over its entire domain $x\in R$ because it is not a one-to-one function (it does not pass the horizontal line test). However, if we restrict the domain to $x\geq 0$ (or $x\leq 0$), an inverse function does exist.

Types of Functions: Exponential Function



Definition: $f(x)=a\times e^{bx}$, where a and b are constants, and e is Euler's number (≈2.71828).

Graph: An upward-sloping curve.

Growth: Rapid and increases as x increases, especially noted for e, which has a base that grows faster than any polynomial but slower than any power of x.

Applications

- **Compound Interest:** Continuously compounded interest is calculated using e.
- **Population Growth:** Often modelled using exponential functions with base e.

Inverse: Logarithmic Function

$$f^{-1}(x) = \frac{1}{b}\ln\left(\frac{x}{a}\right)$$

(natural logarithm)

Recursive Functions

A **recursive function** is a function that calls itself within its own definition. In essence, the function continues to invoke itself, reducing the problem into simpler instances until it reaches a base case which can be solved directly.

One of the recursions is **Fibonacci Numbers**.

Key Elements

Base Case(s): Condition(s) for which the function returns a value directly without further recursion.

Recursive Case(s): Condition(s) where the function calls itself, usually with simpler or reduced input.

Importance in Computing

- Efficiently solve problems with inherent hierarchical or recursive structure.
- Simplify complex problems by breaking them into smaller instances of the same problem.

Example: Recursively defined functions

Assume a recursive function on positive integers:

BC: $f(0) = 3$

RC: $f(n+1) = 2f(n) + 3$

- value of $f(0)$ is 3
- $f(1) = 2f(0) + 3 = 2(3) + 3 = 6 + 3 = 9$
- $f(2) = f(1 + 1) = 2f(1) + 3 = 2(9) + 3 = 18 + 3 = 21$
- $f(3) = f(2 + 1) = 2f(2) + 3 = 2(21) + 3 = 42 + 3 = 45$
- $f(4) = f(3 + 1) = 2f(3) + 3 = 2(45) + 3 = 90 + 3 = 93$

Factorial Function

The *factorial function* is a classic example of a *recursive function*.

The **factorial** of a non-negative integer n, denoted as n!, is the product of all positive integers less than or equal to n. Formally, the factorial function can be defined as:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{if } n > 0 \end{cases}$$

Base Case: 0!=1. The factorial of zero is defined to be 1.
Recursive Step: n!=n×(n−1)! The factorial of n is n times the factorial of n−1.

Example: To find 5!, for instance, we recursively compute:
1!=1×0!; 2!=2×1!; 3!=3×2!; 4!=4×3!; 5!=5×4!