

# Introduction to IT Systems for Beginners

Maciej Gowin @ [CoderBrother](#)

All rights reserved

# Data Formats

- A data format is a **standardized way** to structure and **represent information** for storage, exchange, and processing, enabling systems to communicate effectively.
- Selecting the right data format ensures compatibility, readability, and efficient data handling in applications.

# Data Format: JSON

```
{  
  "firstName": "Maciej",  
  "lastName": "Gowin",  
  "phone": {  
    "callingCode": "48",  
    "number": "693142041"  
  },  
  "height": 180,  
  "dateOfBirth": "1986-11-21",  
  "roles": [  
    "developer",  
    "mentor",  
    "lecturer"  
  ]  
}
```

# Data Format: JSON

- lightweight data-interchange format
- easy for humans to read and write
- easy for machines to parse and generate
- widely used in web development to transmit data between services
- organizes data into key-value pairs
- language agnostic

# Data Format: XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<firstName>Maciej</firstName>
<lastName>Gowin</lastName>
<phone>
  <callingCode>48</callingCode>
  <number>693142041</number>
</phone>
<height>180</height>
<dateOfBirth>1986-11-21</dateOfBirth>
<roles>developer</roles>
<roles>mentor</roles>
<roles>lecturer</roles>
```

# Data Format: XML

- markup language with nested tags to structure data
- hierarchical, extensible, widely used for web services
- language agnostic

# Data Format: CSV

```
Maciej,Gowin,1986-11-21  
Karolina,Kuchna,1989-09-19  
Marian,Nowak,2011-07-04
```

# Data Format: CSV

- comma-Separated Values, used for tabular data
- plain text, row-based, suitable for spreadsheets and data tables
- no nesting, with 2 dimensions

# Data Format: YAML

```
application:  
  name: "Weather Application"  
  version: "1.2.3"  
  
users:  
  user1:  
    name: "gowinm"  
    roles:  
      - developer  
      - mentor  
  user2:  
    name: "nowakm"  
    roles:  
      - tester
```

# Data Format: YAML

- uses indentation for structure, easy to read
- human-friendly, minimal syntax, commonly used in configuration files
- language agnostic

# Data Format: Properties

```
application.name = Weather Application
application.version = 1.2.3
users.user1.name = gowinm
users.user1.roles[0] = developer
users.user1.roles[1] = mentor
users.user2.name = nowakm
users.user2.roles[0] = tester
```

# Data Format: Properties

- simple key-value pairs, used in configuration files
- lightweight, easy to parse, no nesting

# Data Structures

Data structures are ways to organize, manage, and **store data efficiently** for easy access and modification.

They enable efficient data processing and are foundational to algorithms and software development.

## Common Types

Arrays, Lists, Stacks, Queues, Graphs, Trees, Heaps, Maps, and Sets.

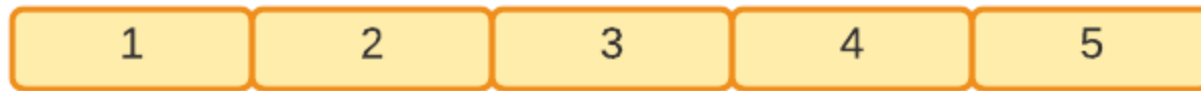
# Data Structure: Array

## Definition

A collection of elements stored in contiguous memory locations, accessible by index.

## Characteristics

- Fixed size, with fast access to elements by index.
- Ideal for scenarios where the number of elements is known and fixed.



# Data Structure: List

## Definition

A sequence of elements that allows dynamic resizing.

## Characteristics

- Can grow or shrink in size.
- Can be implemented as Linked Lists (elements linked by pointers).

## Use Cases

- Dynamic arrays or linked lists.



# Data Structure: Stack

## Definition

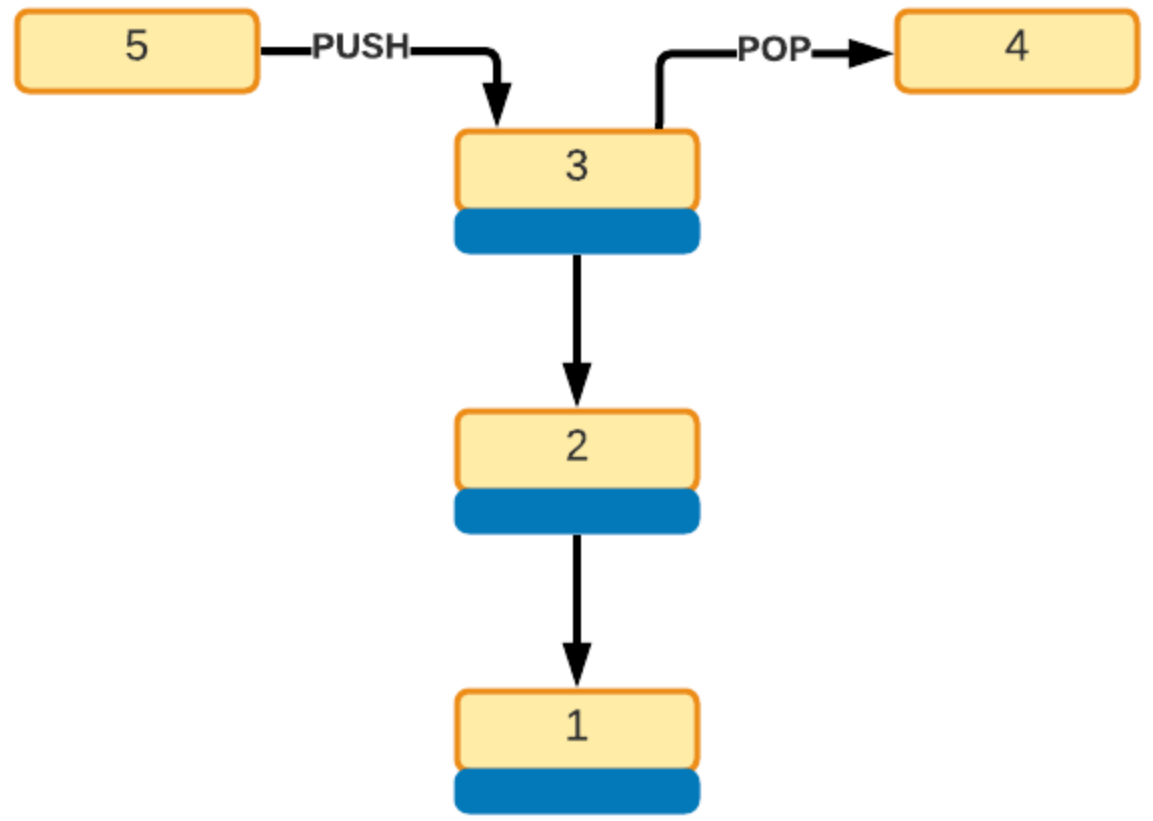
A linear data structure that follows the **Last In, First Out (LIFO)** principle.

## Operations

- **Push:** Add an item to the top.
- **Pop:** Remove the top item.

## Use Cases

- Backtracking, undo mechanisms, and parsing expressions.
- Browser history, function call stack.



# Queue

## Definition

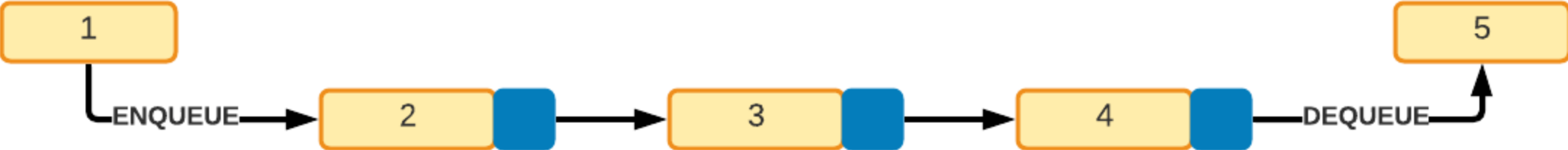
A linear data structure that follows the **First In, First Out (FIFO)** principle.

## Operations

- **Enqueue:** Add an item to the back.
- **Dequeue:** Remove an item from the front.

## Use Cases

- Order processing, task scheduling.
- Print queues, customer service lines.



# Graph

## Definition

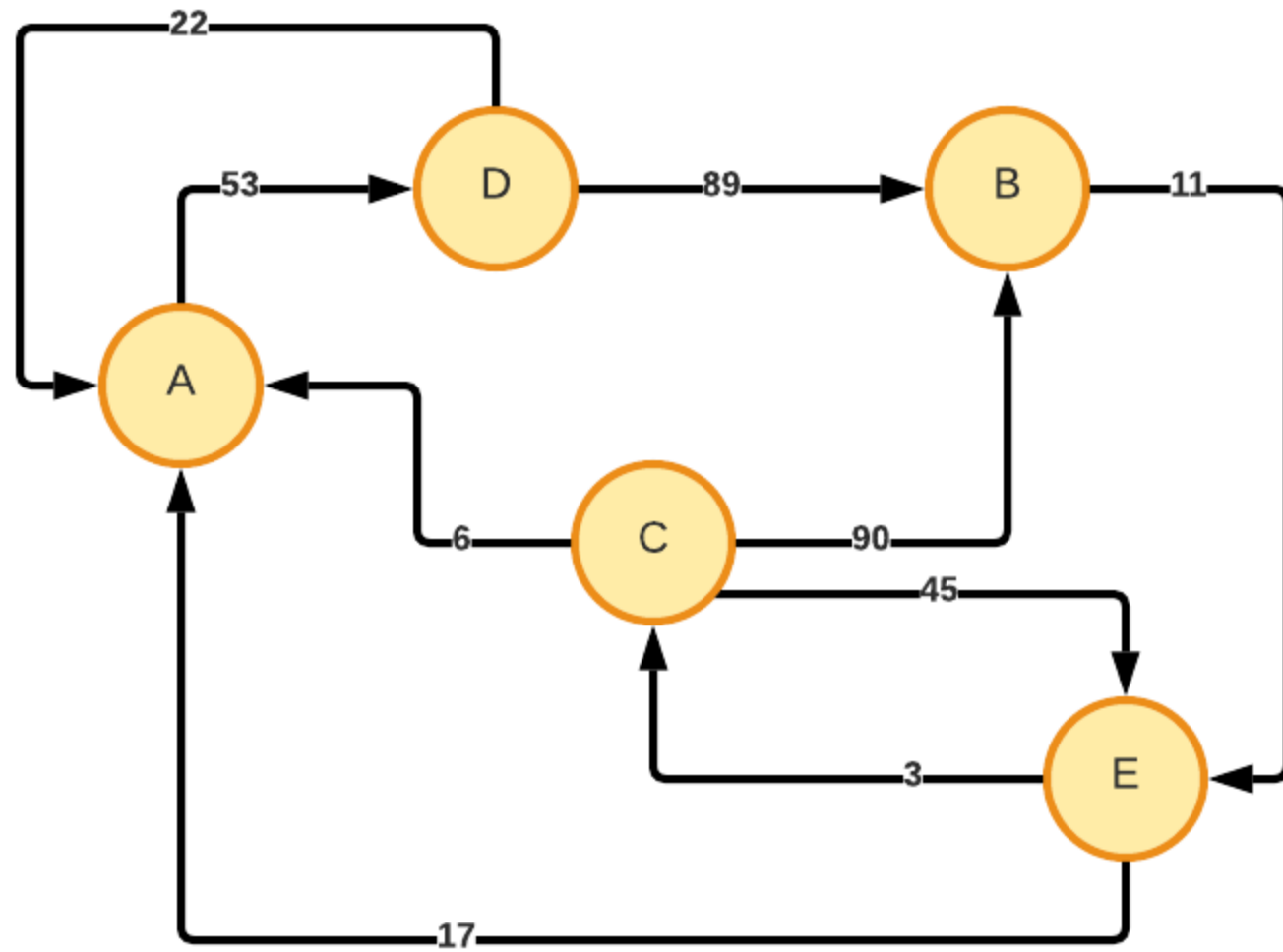
A collection of nodes (vertices) connected by edges.

## Types

- Directed (edges have direction) or Undirected.
- Weighted or Unweighted.

## Use Cases

- Social networks, web page linking, navigation systems.
- Representing cities (nodes) and roads (edges).



# Tree

## Definition

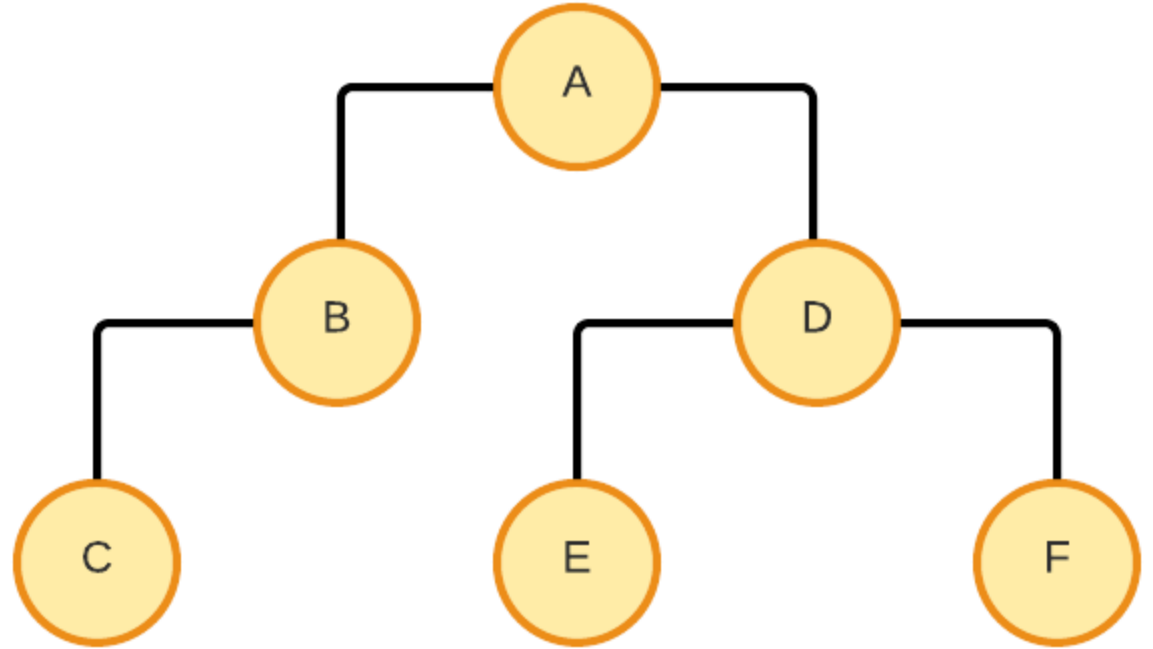
A hierarchical data structure with a root node and child nodes, forming a tree-like structure.

## Types

- Binary Tree (each node has up to 2 children), Binary Search Tree (BST), AVL Tree, etc.

## Use Cases

- Hierarchical data, file systems, databases.
- Organizational chart, XML or JSON document structures.



# Heap

## Definition

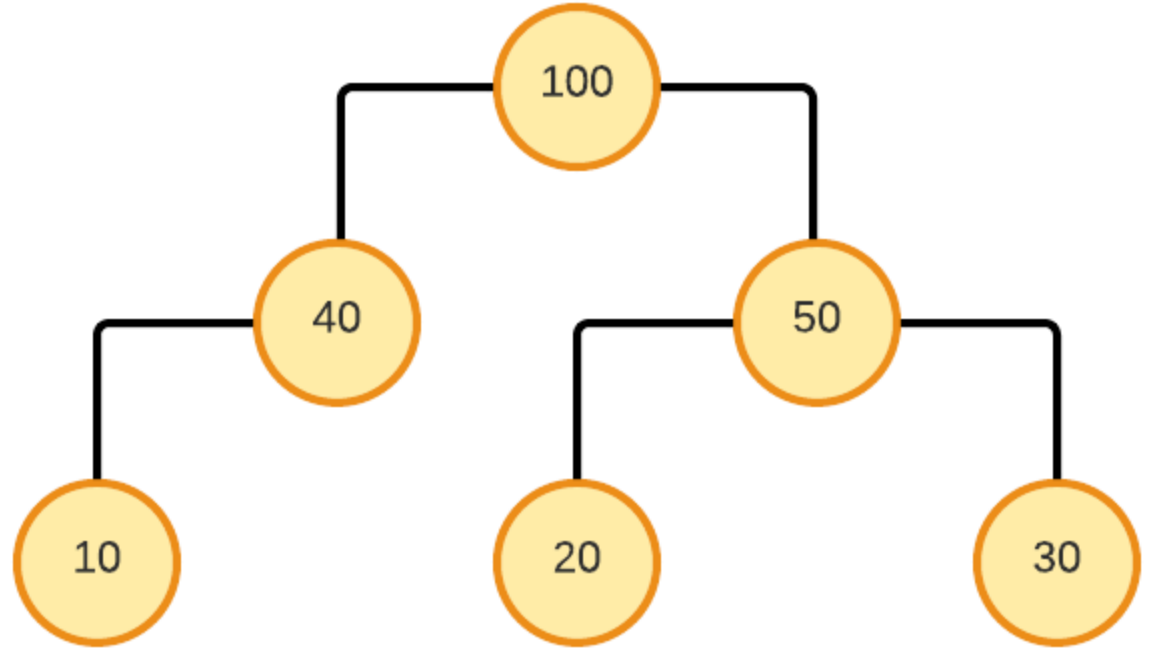
A specialized tree-based structure that satisfies the heap property (min-heap or max-heap).

## Characteristics

- Min-Heap: Root is the minimum value.
- Max-Heap: Root is the maximum value.

## Use Cases

- Priority queues, efficient sorting (heap sort).
- Task schedulers, memory management.



# Map

## Definition

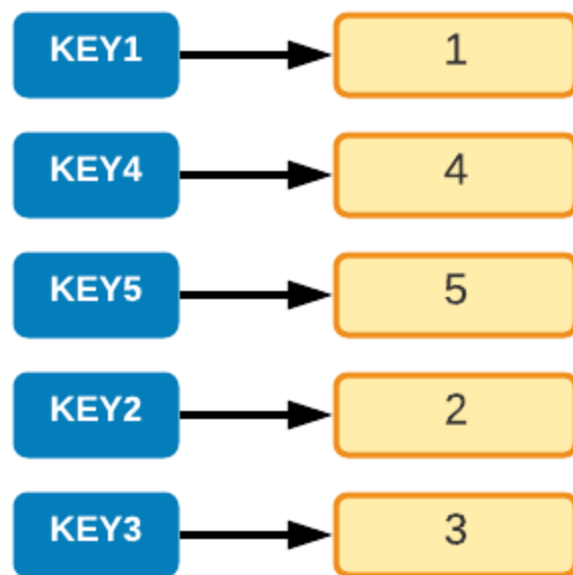
A data structure that stores key-value pairs for fast lookup by key.

## Characteristics

- Each key is unique and maps to one value.

## Use Cases

- Caching, associative arrays, dictionaries.



# Set

## Definition

A collection of unique elements with no specific order.

## Characteristics

- No duplicate elements allowed.
- Often implemented with hash tables for quick lookup.

## Use Cases

- Removing duplicates, membership testing.

1

4

5

2

3

# Algorithm

An **algorithm** is a step-by-step procedure or formula for solving a problem or completing a task.

## Example

A recipe for cooking is an algorithm with instructions to follow step-by-step.

# Characteristics of Algorithm

- **Input:** Data or parameters the algorithm works with.
- **Output:** The result produced by the algorithm.
- **Finite Steps:** Has a clear beginning and end.
- **Efficiency:** Executes tasks in an optimal manner.

# Basic types of Algorithms

## 1. Sorting Algorithms

- Arrange data in a specific order (e.g., ascending or descending).
- Examples: Bubble Sort, Quick Sort, Merge Sort.

## 2. Searching Algorithms

- Find specific data within a collection.
- Examples: Linear Search, Binary Search.

# Basic types of Algorithms

## 3. Graph Algorithms

- Used on graph structures to find paths, shortest routes, or network connections.
- Examples: Dijkstra's Algorithm, Depth-First Search (DFS), Breadth-First Search (BFS).

## 4. Divide and Conquer Algorithms

- Break down a problem into smaller subproblems, solve them, and combine results.
- Example: Merge Sort, Quick Sort.

# Examples of Sorting Algorithms

## Bubble Sort

- Repeatedly compares and swaps adjacent elements to sort a list.
- Easy to understand but not efficient for large datasets.

## Quick Sort

- Uses the divide and conquer approach, selecting a pivot and partitioning elements.
- Faster than Bubble Sort for large datasets.

## Merge Sort

- Divides the list into halves, sorts each half, and merges them.
- Sorting large datasets, combining sorted lists.

# Examples of Searching Algorithms

## Linear Search

- Checks each element one by one until the target is found.
- Simple but slow for large lists.

## Binary Search

- Efficient for sorted lists; repeatedly divides the list in half to find the target.
- Searching for a word in a dictionary, finding an item in a sorted list.

# Algorithm Efficiency: Big O Notation

## What is Big O Notation?

A way to describe an algorithm's efficiency, specifically its time and space complexity.

## Why It Matters:

Helps compare algorithms based on how they perform with large inputs.

## Examples of Complexity

- **$O(1)$** : Constant time; independent of input size.
- **$O(n)$** : Linear time; grows with input size.
- **$O(\log n)$** : Logarithmic time; efficient for large data (e.g., binary search).

# Algorithms by Example

## Problem

The sum of N natural numbers.

## Input/output

1 -> 1

2 -> 3

3 -> 6

# Algorithms by Example: naive

## Solution

Sum all consecutive natural numbers up to N.

## Input/output

**1 -> 1**

**2 -> 1 + 2 = 3**

**3 -> 1 + 2 + 3 = 6**

# Algorithms by Example: optimal

## Solution

Multiply N by N + 1 and divide by 2.

## Input/output

$$1 \rightarrow (1 * (1 + 1))/2 = (1 * 2)/2 = 2/2 = 1$$

$$2 \rightarrow (2 * (2 + 1))/2 = (2 * 3)/2 = 6/2 = 3$$

$$3 \rightarrow (3 * (3 + 1))/2 = (3 * 4)/2 = 12/2 = 6$$

# Brute Force Algorithm

A **brute force algorithm** tries all possible solutions to find the answer. It systematically explores each option until it finds the correct one.

## Examples

- **Password Cracking:** Attempting every possible combination to guess a password.
- **String Matching:** Searching for a substring by comparing every possible position in the main string.
- **Traveling Salesperson Problem:** Checking all possible routes to find the shortest one.

# Characteristics of Brute Force Algorithms

- **Simple and Straightforward:** Often easy to understand and implement.
- **Exhaustive:** Tests all possibilities, which ensures an accurate answer.
- **Inefficient for Large Inputs:** Can be very slow because it doesn't optimize or skip unnecessary steps.
- **High time complexity:** such as  $O(n!)$  or  $O(2^n)$ , making it impractical for large datasets.

# Importance of algorithms

- Algorithms are essential tools for **solving problems efficiently**.
- **The best algorithm** depends on the problem type, data size, and efficiency needs.
- Working with algorithms in different contexts builds **problem-solving skills** and computational thinking.