DEVELOPMENT OF STRUCTIFY: A GAMIFIED INTERACTIVE WEB-BASED APPLICATION FOR DATA STRUCTURES AND ALGORITHMS TAILORED FOR 2ND YEAR BSIT STUDENTS AT PAMANTASAN NG LUNGSOD NG VALENZUELA

LESSONS COVERED BASED ON PLV IT DEPARTMENT DSA SYLLABUS

Weeks	Course Contents	Essential Questions	Intended Learning Outcomes	Assessment Tasks
1st	Course Orientation, Introduction to Basic Data Structures	What is a data structure, and how does it organize data?	Define data structures and standard algorithms.	Pre-assessment Test, Assignment 1
2nd	Preliminaries	How to develop an algorithm in creating and processing data structures?	Describe algorithms and their time/memory complexity.	Problem Solving 1, Assignment 2
3rd	String Processing	How is data stored and processed by a computer?	Understand memory organization and management.	Problem Solving 2, Quiz 1, Assignment 3
4th-5th	Arrays, Records, and Pointers	How are records frequently organized and accessed?	Create, organize, and analyze simple and complex arrays and records.	Problem Solving 2, Program Algorithm 1
6th	Linked Lists	What are linked lists, and how do they differ from arrays?	Explain different operations of arrays and linked lists.	Problem Solving 3, Quiz 2, Assignment 4
7th-8th	Stacks, Queues, Recursion	How are insertion and deletion done in stacks, queues, and recursion?	Restrict data access using stacks, queues, and recursion.	Problem Solving 4, Program Algorithm 2
9th	MIDTERM EXAMINATION			

10th-11th	Trees	How are elements processed through nodes in different tree structures?	Represent hierarchical relationships of elements graphically.	Problem Solving 5, Program Algorithm 3, Assignment 5
12th-13th	Graph Algorithm	How do graphs represent memory, and what are their operations?	Determine graph representations and perform various graph operations.	Problem Solving 6, Quiz 3, Assignment 6
14th-16th	Sorting and Searching	Why are sorting and searching necessary?	Reduce problem complexity using sorting and searching techniques.	Problem Solving 7, Program Algorithm 4, Assignment 6
17th	Hashing	What is hashing, and how does it optimize data retrieval?	Design systems for associative data storage and fast data access.	Problem Solving 8, Quiz 4
18th	Final Project Presentation			

Group Members

- Bautista, Andrew
- Bretaña, Jhonn Michael
- Pantaleon, Kurt Arthur
- Salinas, Julius Louise

Week 1: Introduction to Data Structures

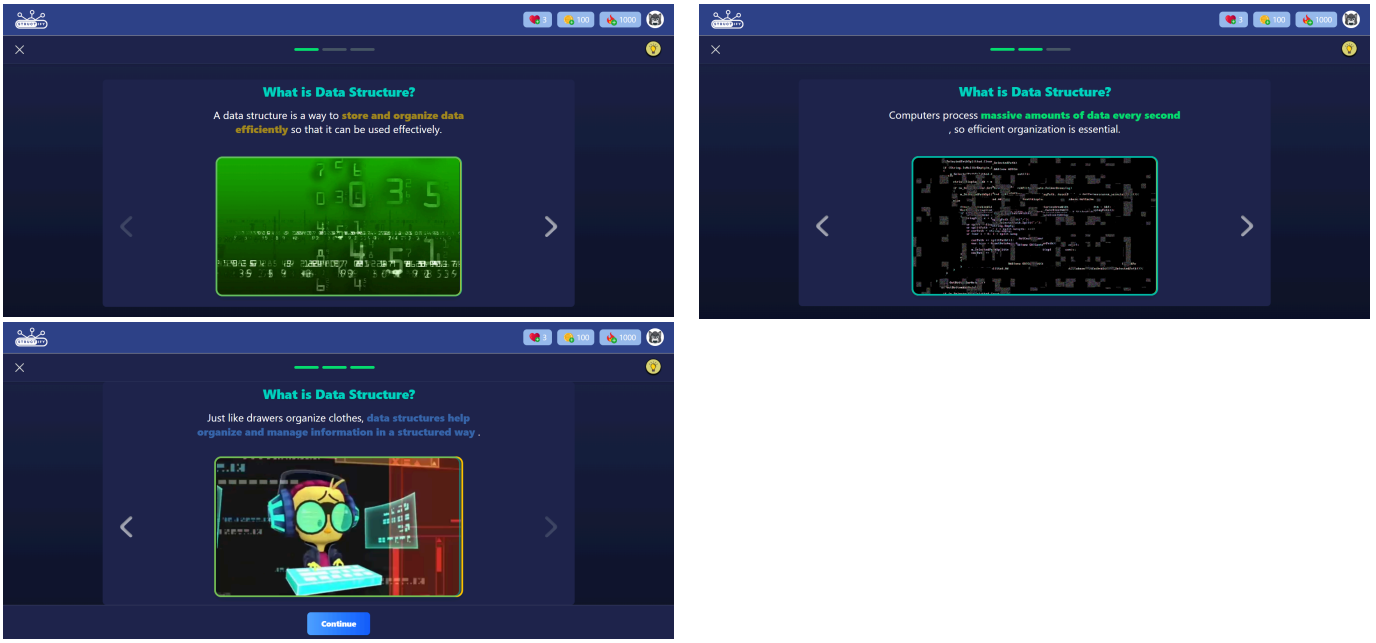
Main Goal:

Understand what data structures are and how they help organize data efficiently.

Lesson 1: What is a Data Structure?

📖 **Concepts to Cover:**

1. **Definition:**
   - A **data structure** is a way to store and organize data efficiently so that it can be used effectively.
   - Computers process massive amounts of data every second, so efficient organization is essential.
   - Just like **drawers** organize clothes, **data structures** help organize and manage information in a structured way.
2. **Why are Data Structures Important?**
   - Without data structures, storing and retrieving information would be **slow and inefficient**.
   - Imagine looking for a book in a **messy pile** vs. a **well-organized bookshelf**.
   - Computers rely on **optimized data structures** to handle searches, sorting, and storage operations efficiently.
3. **How They Improve Efficiency:**
   - **Example:** A **search engine** uses data structures to **index** and find results **instantly**.
   - **Example:** A **video streaming service** uses data structures to **store user preferences** for faster recommendations.
4. **Real-Life Examples of Data Structures:**
   - **Phone Contact List:** Stores names and numbers **alphabetically** for easy lookup.
   - **Bookshelf Organization:** Arranges books **by genre or author** for fast access.
   - **To-Do List:** Prioritizes tasks in a structured way.
5. **Comparison: Messy vs. Well-Organized Data:**
   - **Without structure:** Searching through **scattered** data is slow.
   - **With structure:** Searching through an **indexed** list is fast and efficient.
6. **Common Data Structures:**
   - **Arrays:** Store multiple values in a fixed order.
   - **Linked Lists:** Use nodes to store elements dynamically.
   - **Stacks & Queues:** Follow specific rules for data insertion and removal.
   - **Hash Tables:** Store key-value pairs for fast data retrieval.
7. **When to Use Different Data Structures:**
   - **Fast access?** → Use an **Array**.
   - **Frequent additions/removals?** → Use a **Linked List**.
   - **Undo functionality?** → Use a **Stack** (LIFO).
   - **Processing requests?** → Use a **Queue** (FIFO).
8. **Conclusion:**
   - **Data structures are essential** for building **efficient** and **scalable** applications.

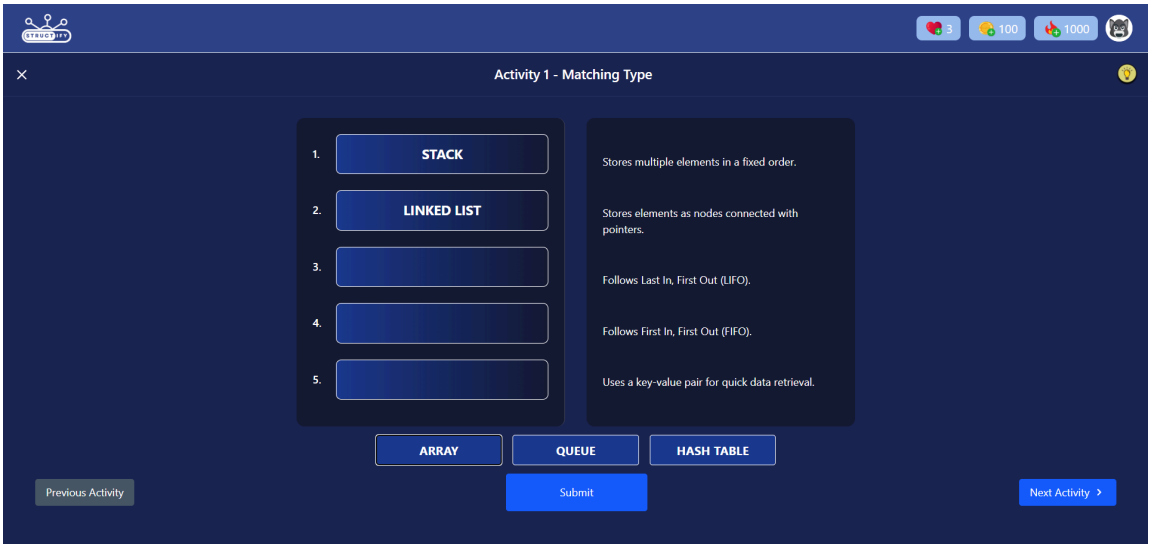


Lesson Pages Example for Structify System

Activity 1: Matching Type

Match the data structure with its description:

- **Array** → Stores multiple elements in a fixed order.
- **Linked List** → Stores elements as nodes connected with pointers.
- **Stack** → Follows Last In, First Out (LIFO).
- **Queue** → Follows First In, First Out (FIFO).
- **Hash Table** → Uses a key-value pair for quick data retrieval.



Activity Page Example for Structify System

Lesson 2: Types of Basic Data Structures

📖 **Concepts to Cover:**

1. **Arrays:**
    - A **list** of items stored in a specific order.
    - Each element has an **index** (position).
    - **Example:** Storing the **top 5 student scores**.
2. **Stacks (LIFO - Last In, First Out):**
    - Works like a **stack of plates** in a cafeteria.
    - The **last item added** is the **first one removed**.
    - **Example:** Browser **back button** history.
3. **Queues (FIFO - First In, First Out):**
    - Works like a **line of people waiting** at a bank.
    - The **first person in line** is served **first**.
    - **Example:** Printer queue.

Activity 2: Fill in the Blanks

1. **Arrays use an _____ to access their elements.** (**Answer:** Index)
2. **A _____ follows LIFO, meaning the last element added is the first to be removed.** (**Answer:** Stack)
3. **A _____ follows FIFO, meaning the first element added is the first to be removed.** (**Answer:** Queue)
4. **A queue is similar to a _____ of people waiting for a bus.** (**Answer:** Line)
5. **A _____ is a key-value data structure for fast lookups.** (**Answer:** Hash Table)

Lesson 3: How Computers Use Data Structures

📖 **Concepts to Cover:**

1. **Computers & Data Structures:** How they rely on structured data.
    - Computers organize data for quick retrieval and processing.
    - Structures like arrays, stacks, and trees improve efficiency.
    - Used in search engines, databases, and AI systems.
2. **File Systems:** How files are stored and accessed efficiently.
    - File systems use hierarchical structures like directories.
    - Indexing (e.g., B-Trees) speeds up searches.
    - Fragmentation affects storage efficiency.
3. **Databases:** How structured data improves searches.
    - Relational (SQL) and NoSQL databases optimize data handling.
    - Indexing (B-Trees, Hashing) speeds up queries.
    - SQL helps manage and retrieve structured data.
4. **Memory Management:** How operating systems use stacks and queues.
    - **Stacks (LIFO):** Used in function calls, undo features.
    - **Queues (FIFO):** Used in task scheduling, buffering.
    - Paging and segmentation manage RAM efficiently.

5. **Algorithm Efficiency:** How choosing the right structure improves performance.
    - Different structures affect speed and memory usage.
    - **Linked lists vs. arrays:** Trade-offs in access speed.
    - Big O notation helps optimize data handling.
6. **Example 1:** Searching for contacts in a phone directory (uses arrays and hashing).
    - **Arrays:** Store names sequentially for lookup.
    - **Hashing:** Maps names to memory for faster access.
    - Hash tables reduce search time to nearly O(1).
7. **Example 2:** Processing tasks in a queue system (like print jobs).
    - Printers use **FIFO queues** for print jobs.
    - OS task schedulers use **priority queues**.
    - Multi-threading improves processing efficiency.
8. **Code Example:**

Activity 3: Drag and Drop

Drag the correct terms (**Array, Stack, Queue, Linked List, Hash Table**) to their corresponding real-life example:

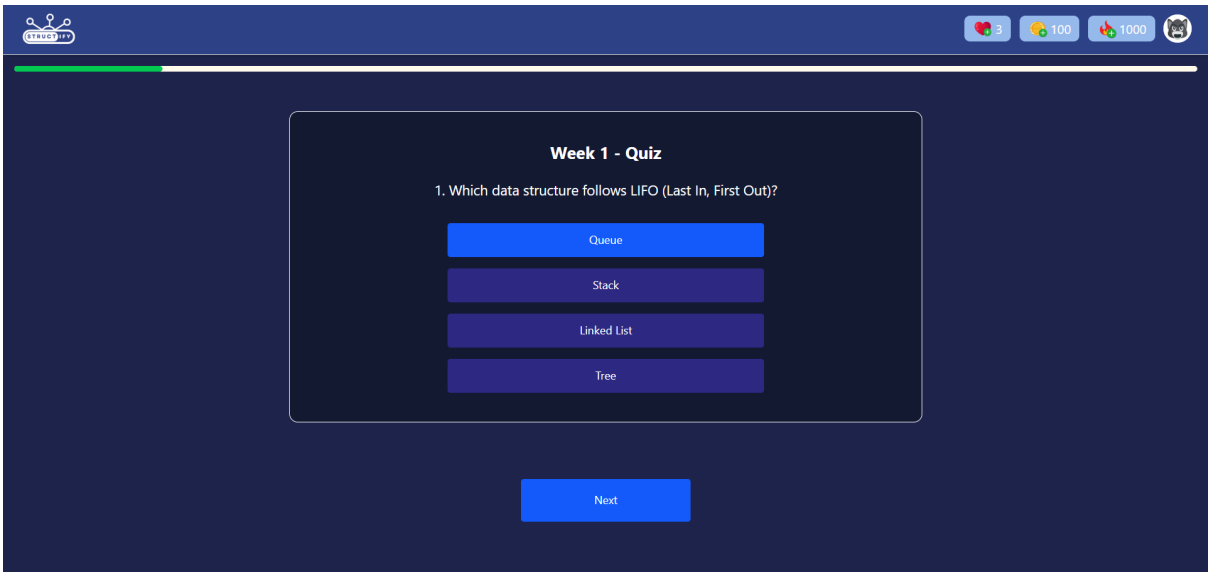
- **Train Line** → Queue
- **Browser Back Button** → Stack
- **Contacts List on Phone** → Array
- **People Standing in Line** → Queue
- **Student Record System** → Hash Table

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. **What is a data structure?**
   A) A way to organize and store data efficiently. ✅ (Correct)
   B) A type of computer processor. ❌ (Wrong)
   C) A file format used for saving data. ❌ (Wrong)
   D) A random collection of numbers. ❌ (Wrong)
   E) A programming language used in web development. ❌ (Wrong)
2. **Why do we need data structures?**
   A) To manage data efficiently and improve performance. ✅ (Correct)
   B) To make computers run slower. ❌ (Wrong)
   C) To delete unnecessary files. ❌ (Wrong)
   D) To increase storage space on a hard drive. ❌ (Wrong)
   E) To design better graphics in games. ❌ (Wrong)
3. **Which data structure follows LIFO (Last In, First Out)?**
   A) Stack ✅ (Correct)
   B) Queue ❌ (Wrong)
   C) Array ❌ (Wrong)
   D) Hash Table ❌ (Wrong)
   E) Record ❌ (Wrong)
4. **What data structure follows FIFO (First In, First Out)?**
   A) Queue ✅ (Correct)
   B) Stack ❌ (Wrong)
   C) Linked List ❌ (Wrong)

D) Hash Table ❌ (Wrong)
E) Array ❌ (Wrong)

5. **A _____ organizes and stores data in a structured way.** (**Answer:** Data Structure)
6. **An _____ stores multiple elements in a fixed order.** (**Answer:** Array)
7. **A _____ is used to map unique keys to specific values for quick retrieval.** (**Answer:** Hash Table)
8. **What happens if data is not organized properly in a program?**
   A) Searching and processing data becomes slow and inefficient. ✅ (Correct)
   B) The computer ignores unorganized data. ❌ (Wrong)
   C) The program automatically arranges the data. ❌ (Wrong)
   D) Data structures will fix themselves over time. ❌ (Wrong)
   E) The program will crash instantly. ❌ (Wrong)



Quiz Page Example for Structify System

Week 2: Preliminaries – Algorithms & Complexity

Main Goal:

Understand how algorithms are created and how they affect time and memory efficiency.

Lesson 1: What is an Algorithm?

📖 **Concepts to Cover:**

1. **Definition:**
    - An **algorithm** is a **step-by-step process** used to solve a problem.
    - It is a **set of instructions** that tell a computer how to accomplish a task.
    - Algorithms are used in **search engines, games, banking systems, and more**.
2. **Examples in Daily Life:**
    - Making a **sandwich** (steps: get bread → add filling → close sandwich).
    - **Tying shoelaces** (steps: cross laces → loop → tighten).
    - **Solving a Rubik's cube** (following specific moves).
3. **Why Do We Need Algorithms?**
    - Computers follow **exact** instructions.
    - Without algorithms, computers **would not know what to do**.
    - They help in **problem-solving, automation, and efficiency**.
4. **Programming Without an Algorithm:**
    - Can lead to **inefficient code, errors, and poor performance**.
    - Example: Sorting numbers randomly instead of using a structured method.
5. **Types of Algorithms:**
    - **Searching Algorithms:** Find elements in data (e.g., Binary Search).
    - **Sorting Algorithms:** Organize elements (e.g., Bubble Sort, Quick Sort).
    - **Encryption Algorithms:** Secure information (e.g., AES, RSA).
    - **Pathfinding Algorithms:** Find shortest paths (e.g., Dijkstra's Algorithm).
6. **Algorithm Structure:**
    - **Input:** The data the algorithm receives.
    - **Processing:** The operations performed on the data.
    - **Output:** The final result after processing.
7. **Real-World Example:**
    - Following a **recipe** is similar to an algorithm for cooking.
    - Searching for a contact in a **phonebook** is like **Binary Search**.
8. **Conclusion:**
    - Algorithms make computing tasks **possible, structured, and efficient**.

Activity 1: Matching Type

Match the term with its correct definition:

- **Algorithm** → A step-by-step process to solve a problem.
- **Efficiency** → How fast an algorithm runs.
- **Big-O Notation** → Measures time complexity.
- **Input** → The data an algorithm receives.
- **Output** → The final result after processing.

Lesson 2: Time & Space Complexity (Efficiency)

📖 **Concepts to Cover:**

1. **Why Efficiency Matters:**
    - **Slow apps and websites** frustrate users.
    - **Faster algorithms** improve speed and performance.
    - Example: A **fast-loading website** vs. a **slow-loading website**.
2. **Big-O Notation:**
    - Describes how an **algorithm's performance** changes as data grows.
    - Helps measure **time complexity** and **space complexity**.
3. **Time Complexity:**
    - Measures **how long an algorithm takes** to run.
    - Example: **Sorting 1,000 vs. 1,000,000 numbers**.
4. **Space Complexity:**
    - Measures **how much memory an algorithm uses**.
    - Some algorithms need **extra memory** (e.g., Merge Sort).
5. **Best, Worst, and Average Case Scenarios:**
    - **Best Case:** Algorithm runs in the least time possible.
    - **Worst Case:** Algorithm runs in the most time possible.
    - **Average Case:** The expected running time.
6. **Example:**
    - Searching for a name in an **organized phone book** (Binary Search).
    - Searching for a name in a **messy pile of papers** (Linear Search).
7. **Real-World Analogy:**
    - **Finding a book in a library** (organized vs. unorganized).
8. **Conclusion:**
    - **Optimizing algorithms saves time, memory, and computing power**.

Activity 2: Fill in the Blanks

1. **An _____ is a set of instructions to solve a problem.** (**Answer:** Algorithm)
2. **_____ tells us how quickly an algorithm runs.** (**Answer:** Efficiency)
3. **The _____ notation helps analyze performance.** (**Answer:** Big-O Notation)
4. **_____ complexity measures the memory used by an algorithm.** (**Answer:** Space)
5. **A slow algorithm takes too much _____ to complete a task.** (**Answer:** Time)

Lesson 3: Writing a Simple Algorithm

📖 **Concepts to Cover:**

1. **Algorithms in Coding:**
    - Instructions **computers follow** to complete tasks.
    - Example: **Sorting a list of numbers**.
2. **Sorting Algorithms:**
    - **Bubble Sort:** Repeatedly swaps adjacent elements.
    - **Selection Sort:** Finds the smallest element and places it first.
    - **Quick Sort:** Uses **pivoting** to divide and sort faster.
3. **Bubble Sort (Simplified):**

    - Compare two numbers, **swap** if needed, **repeat**.
4. **Step-by-Step Example:** Sorting [3, 1, 4, 2].
    - Step 1: [1, 3, 4, 2]
    - Step 2: [1, 3, 2, 4]
    - Step 3: [1, 2, 3, 4]
5. **Comparison-Based Sorting:**
    - **Finding the smallest/largest value** first.
6. **Java Code Example (Bubble Sort):**
7. **Real-Life Analogy:**
    - Sorting **playing cards** in order.
8. **Conclusion:**
    - Sorting algorithms **help organize data efficiently**.

Activity 3: Drag and Drop

Drag the **Big-O complexity class** to its example:

- **O(1)** → Accessing an element in an array.
- **O(n)** → Linear Search.
- **O(log n)** → Binary Search.
- **O(n²)** → Bubble Sort.
- **O(n log n)** → Merge Sort.

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. **What is an algorithm?**
   A) A step-by-step process for solving a problem. ✅ (Correct)
   B) A type of software for designing UI. ❌ (Wrong)
   C) A function used only for sorting data. ❌ (Wrong)
   D) A computer hardware component. ❌ (Wrong)
   E) A method for increasing internet speed. ❌ (Wrong)
2. **Why do computers need algorithms?**
   A) To perform tasks efficiently by following a logical sequence. ✅ (Correct)
   B) To slow down processing speed for security reasons. ❌ (Wrong)
   C) To store large amounts of data without organizing it. ❌ (Wrong)
   D) To make random decisions without human input. ❌ (Wrong)
   E) To change computer hardware settings automatically. ❌ (Wrong)
3. **What does "Big-O Notation" measure?**
   A) The time and space efficiency of an algorithm. ✅ (Correct)
   B) The weight of data stored in memory. ❌ (Wrong)
   C) The maximum file size an algorithm can process. ❌ (Wrong)
   D) The number of variables used in a program. ❌ (Wrong)
   E) The number of programming languages an algorithm supports. ❌ (Wrong)
4. **A _____ is used to describe an algorithm's efficiency.** (**Answer:** Complexity)
5. **Why is a slow algorithm bad?**
   A) It takes more time and resources to complete a task. ✅ (Correct)
   B) It helps users take breaks from computing tasks. ❌ (Wrong)
   C) It makes games more enjoyable. ❌ (Wrong)

D) It speeds up database queries. ❌ (Wrong)
E) It increases internet speed. ❌ (Wrong)

6. **What happens when you reverse "DOG"?** (**Answer:** "GOD")
7. **Which sorting algorithm works by repeatedly swapping adjacent elements?** (**Answer:** Bubble Sort)
8. **Searching for an item in an organized list is _____ than searching randomly.** (**Answer:** Faster)

Week 3: String Processing

Main Goal:

Learn how computers store and process text.

Lesson 1: What are Strings?

📖 **Concepts to Cover:**

1. **Definition:**
   - A **string** is a **sequence of characters** used to store text.
   - Strings can include **letters, numbers, spaces, and symbols**.
   - Strings are enclosed in **double quotes ("Hello")** or **single quotes ('Hello')** in most programming languages.
2. **Examples of Strings:**
   - "Hello, World!" → A **common string example** in programming.
   - "Kurt" → A **person's name** stored as a string.
   - "1234" → A **number stored as a string**, meaning it is treated as text, not a numerical value.
3. **Why Do We Need Strings?**
   - Strings allow computers to **display messages, store usernames, and process text data**.
   - **Example:** A **chatbot** processes user input **as a string** before responding.
4. **Strings vs. Numbers:**
   - "123" (String) ≠ 123 (Integer).
   - Strings **cannot be used for arithmetic operations** unless explicitly converted.
5. **Real-World Usage:**
   - **Usernames and passwords** on websites.
   - **File names** stored in an operating system.
   - **Messages** sent in chat applications.
6. **String Formatting:**
   - Strings can be **combined (concatenated)** or **formatted** to create structured sentences.
   - **Example:**
7. **Mutable vs. Immutable Strings:**
   - In **Java**, **strings are immutable** (cannot be changed after creation).
   - Any modification creates **a new string object** instead of modifying the original.
8. **Conclusion:**

- Strings are **a fundamental part of data storage** and widely used in programming, databases, and file systems.

Activity 1: Matching Type

Match the function with its purpose:

- **toUpperCase()** → Converts text to uppercase.
- **toLowerCase()** → Converts text to lowercase.
- **replace()** → Replaces a substring.
- **trim()** → Removes extra spaces at the beginning and end.
- **length()** → Counts the number of characters in a string.

Lesson 2: String Operations

📖 **Concepts to Cover:**

1. **String Concatenation (Joining Strings):**
   - Combining multiple strings using + in Java.
   - **Example:**
2. **String Indexing (Accessing Characters):**
   - You can **access specific characters** using charAt(index).
   - **Example:**
3. **String Slicing (Substring Extraction):**
   - Extract a portion of a string using substring(start, end).
   - **Example:**
4. **Reversing a String (Using a Loop):**
5. **Common String Methods:**
   - **toUpperCase()** → "hello".toUpperCase() → "HELLO"
   - **toLowerCase()** → "HELLO".toLowerCase() → "hello"
   - **replace()** → "hello".replace("l", "z") → "hezzo"
   - **split()** → "apple,banana".split(",") → ["apple", "banana"]
6. **Finding Text in a String:**
   - Use indexOf() to find **the position of a character**.
   - **Example:**
7. **Using Escape Characters:**
   - \n → New line.
   - \t → Tab space.
   - \\ → Backslash.
8. **Conclusion:**
   - String operations **allow text manipulation** in programs.

Activity 2: Fill in the Blanks

1. **The _____ function converts text to uppercase.** (**Answer:** toUpperCase())
2. **_____ breaks a string into smaller parts.** (**Answer:** split())
3. **To replace "hello" with "hi", use _____.** (**Answer:** replace())
4. **The function to remove spaces at the beginning and end is _____.** (**Answer:** trim())
5. **To count the length of a string, use _____.** (**Answer:** length())

Lesson 3: How Computers Process Text

📖 **Concepts to Cover:**

1. **How Computers Store Text:**
   - Uses **ASCII** (American Standard Code for Information Interchange) and **Unicode**.
2. **ASCII Codes:**
   - Each character has a unique number.
   - **Example:** "A" = 65, "B" = 66.
3. **Unicode Encoding:**
   - Supports **multiple languages and emojis**.
   - "😃" is stored as **U+1F603**.
4. **Binary Representation of Text:**
   - Computers **store text as binary (0s and 1s)**.
5. **Why Encoding Matters:**
   - Prevents **text corruption** between different systems.
6. **Real-World Example:**
   - Websites displaying "" when a character is unsupported.
7. **Text is Data!**
   - Emails, documents, and **social media posts** are all stored as strings.
8. **Conclusion:**
   - Encoding helps computers **process text efficiently worldwide**.

Activity 3: Drag and Drop

Drag the correct output for each string operation:

- "hello".toUpperCase() → "HELLO"
- "HELLO".toLowerCase() → "hello"
- "apple,banana,orange".split(",") → ["apple", "banana", "orange"]
- " hello ".trim() → "hello"
- "Python".length() → 6

Weekly Quiz

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. **What is a string?**
2. **Why do computers need strings?**
3. **What is ASCII used for?**
4. **The function to remove spaces at the beginning and end of a string is _____.** (**Answer:** trim())
5. **How do you make text lowercase in Java?**
6. **What's the ASCII code for "A"?** (**Answer:** 65)
7. **What happens when you reverse "dog"?** (**Answer:** "god")
8. **Computers store text using _____ encoding.** (**Answer:** ASCII/Unicode)

Week 4-5: Arrays, Records, and Pointers

Main Goal:

Learn how computers organize and access data efficiently using arrays, records, and pointers.

Lesson 1: Understanding Arrays – The Basics

📖 **Concepts to Cover:**

1. **What is an Array?**
    - An **array** is a **collection of elements** stored in a fixed order.
    - It holds **multiple values** of the **same data type**.
2. **Real-Life Analogy:**
    - Think of an array like a **row of lockers** where each locker has a **unique number** (index).
    - The first locker is **index 0**, the second is **index 1**, and so on.
3. **Why Use Arrays?**
    - Arrays allow **efficient storage and retrieval** of multiple values.
    - Instead of creating separate variables, you can store **all related values in one structure**.
4. **Examples of Arrays:**
    - A **list of your favorite video games**.
    - A **shopping cart** in an e-commerce app.
    - A **leaderboard in a game** tracking high scores.
5. **Declaring an Array in Java:**
6. **Accessing Elements:**
    - Arrays use **indexing** to retrieve data.
    - games[0] retrieves the **first element** ("Minecraft").
7. **Modifying an Array:**
8. **Conclusion:**
    - Arrays **store, access, and modify** multiple values efficiently.

Activity 1: Matching Type

Match the data type with its correct use case:

- **Array** → Stores elements in order.
- **Record** → Stores multiple data types.
- **Pointer** → Stores memory addresses.

Lesson 2: Understanding Records – Grouping Data

📖 **Concepts to Cover:**

1. **What is a Record?**
    - A **record** is a **collection of related data** grouped together.
    - Unlike arrays, **each field can have a different data type**.
2. **Real-Life Analogy:**
    - A **contact list** where each person has a **name, phone number, and email**.
3. **Why Use Records?**
    - They allow **structured storage** of different types of information.
    - Common in **databases** and **applications** that store user profiles.
4. **Declaring a Record in Java (Using a Class):**
5. **How Databases Use Records:**
    - Databases store **user information** as records.
    - **Example:** A **student record system** storing names, grades, and attendance.
6. **Records vs. Arrays:**
    - **Arrays** store **multiple values of the same type**.
    - **Records** store **multiple related fields of different types**.
7. **Example of Using an Array of Records:**
8. **Conclusion:**
    - Records provide a **structured way** to store related **but different types of data**.

Activity 2: Fill in the Blanks

1. **An _____ holds multiple elements of the same type.** (**Answer:** Array)
2. **A _____ stores a group of related data with different types.** (**Answer:** Record)
3. **A _____ points to a memory location.** (**Answer:** Pointer)

Lesson 3: Pointers – How Computers Track Data

📖 **Concepts to Cover:**

1. **What are Pointers?**
    - A **pointer** is a variable that **stores the memory address** of another variable.
    - Instead of storing a value, it **"points"** to where the value is stored in memory.
2. **Real-Life Analogy:**
    - A **bookmark in a book** remembers your place without copying the entire book.
3. **Why Do Computers Use Pointers?**
    - Instead of **copying data**, pointers **reference existing data**.
    - They make **programs more efficient** by reducing **memory usage**.
4. **Declaring and Using Pointers in Java (Using Object References):**
5. **How Pointers Help in Memory Management:**
    - **Dynamic Memory Allocation:** Allocates memory only when needed.
    - **Efficient Data Structures:** Used in **linked lists** and **trees**.
6. **Example of Using Pointers with Objects:**
7. **Pointers vs. Regular Variables:**
    - **Regular variables** store values directly.
    - **Pointers** store **memory addresses** of those values.

8. **Conclusion:**
    - Pointers **improve efficiency** and **help manage memory dynamically**.

Activity 3: Drag and Drop

Drag the correct data type to the example:

- **Shopping List** → Array
- **Student Record** → Record
- **Memory Reference** → Pointer

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. **What is an array?**
   A) A collection of elements stored in a fixed order. ✅ (Correct)
   B) A function that repeats indefinitely. ❌ (Wrong)
   C) A random collection of data that changes order. ❌ (Wrong)
   D) A file format for saving text documents. ❌ (Wrong)
   E) A tool for measuring distances in programming. ❌ (Wrong)
2. **How does a record differ from an array?**
   A) A record can store multiple data types, while an array stores a single type. ✅ (Correct)
   B) A record can only store numbers, while an array stores only text. ❌ (Wrong)
   C) Arrays store data with named fields, while records use index positions. ❌ (Wrong)
   D) Records are faster than arrays in all cases. ❌ (Wrong)
   E) Arrays are always larger than records. ❌ (Wrong)
3. **A _____ holds multiple elements of the same type.** (**Answer:** Array)
4. **A _____ stores a group of related data with different types.** (**Answer:** Record)
5. **A _____ points to a memory location.** (**Answer:** Pointer)
6. **What is a pointer used for?**
   A) Storing the memory address of another variable. ✅ (Correct)
   B) Sorting a list of numbers in ascending order. ❌ (Wrong)
   C) Increasing computer speed. ❌ (Wrong)
   D) Organizing data in an array. ❌ (Wrong)
   E) Printing text in a document. ❌ (Wrong)
7. **What does a pointer reference?** (**Answer:** A memory address)
8. **Which data type is used for storing multiple records of student names and grades?** (**Answer:** Record)

Week 6: Linked Lists

Main Goal:

Learn how linked lists work and how they differ from arrays.

Lesson 1: What is a Linked List?

📖 **Concepts to Cover:**

1. **Definition:**
    - A **linked list** is a **data structure** that stores elements in **nodes**.
    - Each **node** contains **data** and a **reference (link) to the next node**.
2. **Real-Life Analogy:**
    - A **treasure hunt**, where each clue leads to the next location.
    - A **train**, where each coach is linked to the next.
3. **How Linked Lists Differ from Arrays:**
    - **Arrays:** Stored in **continuous memory locations**.
    - **Linked Lists:** Use **dynamic memory allocation** and store elements **anywhere in memory**.
4. **Advantages of Linked Lists:**
    - **Dynamic Size:** Can grow or shrink as needed.
    - **Efficient Insertions/Deletions:** No need to shift elements.
5. **Disadvantages of Linked Lists:**
    - **Slower Access:** Must **traverse** the list to find an element.
    - **More Memory Usage:** Requires extra storage for **pointers**.
6. **Example of a Linked List in Java:**
7. **Conclusion:**
    - Linked lists are **flexible**, **efficient for insertions**, and **widely used in programming**.

Activity 1: Matching Type

Match the term with its correct definition:

- **Node** → An element storing data and a reference.
- **Head** → The first node in a linked list.
- **Tail** → The last node, pointing to NULL.

Lesson 2: Nodes and Links

📖 **Concepts to Cover:**

1. **What is a Node?**
    - A **node** is the **basic unit** of a linked list.
    - Each **node** has:
        - **Data** (Value stored in the node).

        - **Next Pointer** (Link to the next node).
2. **How Nodes Connect:**
    - Each **node stores a reference** to the next node.
    - The **last node (Tail)** has a **NULL reference**.
3. **Linked List Structure:**
4. **Adding Nodes to a Linked List:**
    - New nodes can be **dynamically inserted**.
5. **Advantages of Nodes:**
    - **Efficient Insertions:** No need to shift elements like in an array.
    - **Memory Usage:** Can allocate dynamically.
6. **Traversing a Linked List:**
    - **Looping through each node** to access elements.
7. **Real-Life Analogy:**
    - Think of **people in a queue** where each person knows **only the next person**.
8. **Conclusion:**
    - Linked lists are **useful for dynamic memory allocation** and **fast insertions/deletions**.

Activity 2: Fill in the Blanks

1. **A _____ is an element in a linked list.** (**Answer:** Node)
2. **The _____ is the first node in the list.** (**Answer:** Head)
3. **The _____ is the last node, usually pointing to NULL.** (**Answer:** Tail)

Lesson 3: Operations on a Linked List

📖 **Concepts to Cover:**

1. **Adding a Node at the Beginning:**
    - **Time Complexity: O(1)**
2. **Adding a Node at the End:**
    - **Time Complexity: O(n)**
3. **Deleting a Node:**
    - **Time Complexity: O(n)**
4. **Searching in a Linked List:**
5. **Advantages Over Arrays:**
    - **No need for a fixed size.**
    - **Insertion/Deletion is faster.**
6. **Disadvantages Over Arrays:**
    - **Slower access time (O(n))** compared to arrays (**O(1) for index-based access**).
7. **Real-Life Usage:**
    - **Undo/Redo in text editors.**
    - **Music playlists (songs linked to each other).**
8. **Conclusion:**
    - Linked lists are **great for dynamic data storage and memory-efficient operations**.

Activity 3: Drag and Drop

Drag the correct term to its corresponding description:

- **Insertion at Start** → O(1) Complexity
- **Insertion at End** → O(n) Complexity
- **Traversal** → Visiting each node in the list

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. **How does a linked list store data?**
   A) It stores data in nodes, each linking to the next node. ✅ (Correct)
   B) It stores all data in a single continuous memory block. ❌ (Wrong)
   C) It saves data in a compressed format. ❌ (Wrong)
   D) It uses only numerical values to represent information. ❌ (Wrong)
   E) It arranges data in a table format. ❌ (Wrong)
2. **A _____ is the basic unit of a linked list.** (**Answer:** Node)
3. **The _____ is the first node in a linked list.** (**Answer:** Head)
4. **Why do programmers use linked lists?**
   A) Because they allow dynamic memory allocation and efficient insertions/deletions. ✅ (Correct)
   B) Because they consume less memory than arrays in all cases. ❌ (Wrong)
   C) Because they make programs slower on purpose. ❌ (Wrong)
   D) Because they are easier to sort than arrays. ❌ (Wrong)
   E) Because they are the only way to store data in a program. ❌ (Wrong)
5. **What does the tail of a linked list point to?** (**Answer:** NULL)
6. **What is an advantage of linked lists over arrays?** (**Answer:** Dynamic memory allocation)
7. **Which data structure does NOT use fixed memory allocation?** (**Answer:** Linked List)
8. **How do nodes connect in a linked list?** (**Answer:** Using pointers or references)

Week 7-8: Stacks, Queues, and Recursion

Main Goal:

Understand how **stacks, queues, and recursion** work in programming, their real-world applications, and how they help solve computational problems.

Lesson 1: What is a Stack?

📖 **Concepts to Cover (8 Sections):**

1. Definition

- A **stack** is a **linear data structure** that follows the **LIFO (Last In, First Out)** principle.
- The **last item added** is the **first one to be removed**.

2. Real-Life Analogy

- Think of a **stack of plates** in a cafeteria—**you remove the topmost plate first** before accessing the others.
- Undo/Redo in a text editor follows the **stack mechanism**.

3. Basic Operations of a Stack

- **Push:** Adds an element to the stack.
- **Pop:** Removes the top element from the stack.
- **Peek:** Retrieves the top element **without removing** it.
- **isEmpty:** Checks if the stack is empty.

4. Stack Example in Java

5. Stack in Function Calls (Call Stack)

- Every time a function is called, **a new frame is pushed onto the call stack**.
- When the function completes, the **frame is popped** from the stack.

6. Applications of Stacks

- **Undo/Redo operations** in text editors.
- **Backtracking (e.g., navigating browser history).**
- **Expression evaluation (e.g., converting infix to postfix).**

7. Stack Overflow

- If a stack grows beyond its memory capacity, it results in a **Stack Overflow Error**.

8. Conclusion

- Stacks provide **an efficient way** to handle sequential operations where **order matters**.

Activity 1: Matching Type

Match the **data structure** with its characteristic:

- **Stack** → LIFO (Last In, First Out).
- **Queue** → FIFO (First In, First Out).
- **Recursion** → A function calling itself.

Lesson 2: What is a Queue?

📖 **Concepts to Cover (8 Sections):**

1. Definition

- A **queue** is a **linear data structure** that follows the **FIFO (First In, First Out)** principle.
- The **first item added** is the **first one to be removed**.

2. Real-Life Analogy

- Think of a **line at a fast-food counter**—the first person in line **is served first**.

3. Basic Operations of a Queue

- **Enqueue:** Adds an element to the back of the queue.
- **Dequeue:** Removes an element from the front of the queue.
- **Peek:** Retrieves the front element **without removing** it.
- **isEmpty:** Checks if the queue is empty.

4. Queue Example in Java

5. Types of Queues

- **Simple Queue:** FIFO (First-In-First-Out).
- **Circular Queue:** The last position connects back to the first position.
- **Priority Queue:** Elements are dequeued based on priority instead of order.

6. Applications of Queues

- **Task scheduling** (e.g., process scheduling in operating systems).
- **Network packet processing** (e.g., handling multiple requests).
- **Print queue in printers** (serves print requests in order).

7. Queue Overflow & Underflow

- **Overflow:** When a queue is full and no more elements can be added.
- **Underflow:** When a queue is empty and no elements can be dequeued.

8. Conclusion

- Queues are essential for managing data in a **sequential and ordered manner**.

Activity 2: Fill in the Blanks

1. **The _____ data structure is used for undo/redo functions.** (**Answer:** Stack)
2. **A _____ is used for waiting lines.** (**Answer:** Queue)
3. **_____ occurs when a function keeps calling itself.** (**Answer:** Recursion)

Lesson 3: Understanding Recursion

📖 **Concepts to Cover (8 Sections):**

1. Definition

- **Recursion** occurs when a **function calls itself** until it reaches a stopping condition.

2. Real-Life Analogy

- **Russian nesting dolls**—each doll contains a smaller version of itself.

3. Recursion Example in Java

4. Recursive vs. Iterative Approach

- **Iteration (Looping)** runs a loop until the condition is false.
- **Recursion** solves problems **by breaking them into smaller subproblems**.

5. Base Case & Recursive Case

- **Base Case:** Stops the recursion when a condition is met.
- **Recursive Case:** Calls itself with a smaller problem.

6. Applications of Recursion

- **Solving mathematical problems** (e.g., factorial, Fibonacci sequence).
- **Tree and graph traversals** (e.g., file system navigation).
- **Backtracking algorithms** (e.g., maze solving, Sudoku solving).

7. Recursion Limitations

- **Consumes more memory than iteration** due to function calls.
- Can result in a **Stack Overflow Error** if there's no base case.

8. Conclusion

- Recursion simplifies problems **by breaking them into smaller parts** but must be used efficiently.

Activity 3: Drawing Recursive Patterns

- Draw a **large box**, then **a smaller box inside**, repeating the process.
- Explain how this represents **recursion** (each box contains a **smaller version of itself**).

Weekly Quiz: Stacks, Queues, and Recursion

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. **What is a stack?**
   A) A data structure that follows the Last In, First Out (LIFO) principle. ✅ (Correct)
   B) A structure that removes the first item added. ❌ (Wrong)
   C) A type of loop used in programming. ❌ (Wrong)
   D) A method for sorting numbers in ascending order. ❌ (Wrong)
   E) A type of computer memory that permanently stores data. ❌ (Wrong)
2. **What is a queue?**
   A) A data structure that follows the First In, First Out (FIFO) principle. ✅ (Correct)
   B) A method for sorting elements from largest to smallest. ❌ (Wrong)
   C) A storage system that allows random access to elements. ❌ (Wrong)
   D) A structure where the last item added is removed first. ❌ (Wrong)
   E) A function that deletes elements in reverse order. ❌ (Wrong)
3. **How does recursion work?**
   A) A function keeps calling itself until it reaches a stopping condition. ✅ (Correct)
   B) A function runs continuously without stopping. ❌ (Wrong)
   C) The program executes multiple functions at the same time. ❌ (Wrong)
   D) A function stops after a single execution. ❌ (Wrong)
   E) Recursion is only used for numerical calculations. ❌ (Wrong)
4. **A _____ data structure is used in undo/redo functions.** (**Answer:** Stack)
5. **A queue follows the _____ principle.** (**Answer:** FIFO)
6. **Which data structure is used for waiting lines in banks?** (**Answer:** Queue)
7. **Which of the following is a real-life example of recursion?**
   A) Russian nesting dolls, where each doll contains a smaller version of itself. ✅ (Correct)
   B) A person standing in a line at a supermarket. ❌ (Wrong)
   C) A stack of plates in a cafeteria. ❌ (Wrong)
   D) A queue of customers waiting at a bank. ❌ (Wrong)
   E) A book placed inside a drawer. ❌ (Wrong)
8. **Recursion is commonly used in _____ algorithms.** (**Answer:** Tree and graph traversal)

Week 10-11: Trees

Main Goal:

Understand how **tree data structures** work, how they organize elements hierarchically, and their applications in searching, AI, and databases.

Lesson 1: Introduction to Trees

📖 **Concepts to Cover (8 Sections):**

1. Definition

- A **tree** is a **non-linear data structure** that organizes elements **hierarchically**.
- Unlike arrays and linked lists, **trees do not store data sequentially**.

2. Real-Life Analogy

- **A family tree:** Each person (node) is connected to their parents and children.
- **A file system on a computer:** Folders can contain subfolders and files.

3. Tree Terminology

- **Root:** The **top node** of a tree.
- **Parent & Child Nodes:** Nodes are connected in a **parent-child relationship**.
- **Leaf Node:** A node **without children**.
- **Subtree:** A smaller tree that is part of a larger tree.

4. Properties of Trees

- Trees **do not contain cycles** (unlike graphs).
- Every **child node** has only **one parent**.

5. Why Use Trees?

- **Faster searching** than lists and arrays.
- **Organizes data efficiently** (e.g., file systems, databases).

6. Example of a Simple Tree in Java

7. Applications of Trees

- **Used in search engines** to organize web pages.
- **Used in artificial intelligence (AI)** for decision-making.
- **Used in databases** to organize records efficiently.

8. Conclusion

- Trees provide a **flexible and efficient way** to organize and search data.

Activity 1: Matching Type

Match the **tree term** with its definition:

- **Root** → The **top node**.
- **Leaf Node** → A node **with no children**.
- **Binary Tree** → Each node **has at most two children**.

Lesson 2: Binary Trees and Binary Search Trees (BST)

📖 **Concepts to Cover (8 Sections):**

1. What is a Binary Tree?

- A **binary tree** is a **tree where each node has at most two children**.

2. Real-Life Analogy

- **A decision-making process:** At each step, there are two options (left or right).

3. What is a Binary Search Tree (BST)?

- A **BST** is a special binary tree where:
  - **Smaller values** go to the **left subtree**.
  - **Larger values** go to the **right subtree**.

4. Why Are BSTs Useful?

- **Efficient searching:** Finding an element in a BST is **faster than searching in a list**.

5. Insertion in a BST (Java Example)

6. Advantages of BSTs Over Lists

- **Searching is O(log n),** compared to **O(n) in lists**.
- **Sorting is easier** since elements are **already ordered**.

7. When Not to Use BSTs?

- **If the tree is unbalanced,** searching can become slow.
- **Self-balancing trees** (e.g., AVL, Red-Black trees) **solve this issue**.

8. Conclusion

- BSTs **improve efficiency** in searching, sorting, and organizing hierarchical data.

Activity 2: Fill in the Blanks

1. **The _____ is the starting point of a tree.** (**Answer:** Root)
2. **A _____ has no child nodes.** (**Answer:** Leaf Node)

Lesson 3: Tree Traversal (How to Search a Tree)

📖 **Concepts to Cover (8 Sections):**

1. What is Tree Traversal?

- **Traversal** means **visiting all nodes** in a tree in a specific order.

2. Types of Tree Traversal

- **Inorder (Left, Root, Right)**
- **Preorder (Root, Left, Right)**
- **Postorder (Left, Right, Root)**

3. Real-Life Analogy

- **Visiting rooms in a building systematically** to ensure no room is left unchecked.

4. Tree Traversal Example in Java

5. Applications of Tree Traversal

- **Used in AI** for decision trees.
- **Used in compilers** to parse expressions.

6. Searching in BSTs

- **Binary search algorithm** makes lookups **fast and efficient**.

7. Tree-Based Data Structures in Real Life

- **Trie (used in search engines)**.
- **B-Trees (used in databases)**.

8. Conclusion

- Different traversal methods **help computers process hierarchical data efficiently**.

Activity 3: Tree Treasure Hunt

- Create a **decision tree** where students **choose paths to reach an answer**.

📌 **Week 10-11 Quiz: Trees**

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. What is a tree in computer science?

A) A hierarchical data structure where each node connects to child nodes. ✅ (Correct)
B) A type of loop that runs until a condition is met. ❌ (Wrong)
C) A program that generates random numbers. ❌ (Wrong)
D) A method for storing numbers in ascending order. ❌ (Wrong)
E) A mathematical function for sorting values. ❌ (Wrong)

2. What is the topmost node of a tree called?

A) **Root Node** ✅ (Correct)
B) Parent Node ❌ (Wrong)
C) Leaf Node ❌ (Wrong)
D) Branch Node ❌ (Wrong)
E) Trunk Node ❌ (Wrong)

3. How does a Binary Search Tree (BST) organize elements?

A) **Smaller values go to the left subtree, larger values go to the right.** ✅ (Correct)
B) Larger values go to the left subtree, smaller values go to the right. ❌ (Wrong)
C) All values are stored in a straight line. ❌ (Wrong)
D) Elements are stored randomly, with no rules. ❌ (Wrong)
E) The first number is always the largest, and everything follows in descending order. ❌ (Wrong)

4. Which tree traversal method visits the root node first, then left and right subtrees?

A) **Preorder Traversal (Root, Left, Right)** ✅ (Correct)
B) Inorder Traversal (Left, Root, Right) ❌ (Wrong)
C) Postorder Traversal (Left, Right, Root) ❌ (Wrong)
D) Breadth-First Search ❌ (Wrong)
E) Depth-First Search ❌ (Wrong)

5. A _____ node has no children. (Answer: Leaf)

6. The _____ is the starting point of a tree. (Answer: Root)

7. Why are BSTs more efficient than arrays for searching?

A) **Searching is faster because of the sorted structure (O(log n) time complexity).** ✅ (Correct)
B) BSTs use less memory than arrays in all cases. ❌ (Wrong)
C) BSTs automatically delete duplicate values. ❌ (Wrong)
D) BSTs store all data in a single variable. ❌ (Wrong)
E) BSTs are easier to write in every programming language. ❌ (Wrong)

8. How do search engines use trees?

A) **They organize web pages in a structured hierarchy to improve search efficiency.** ✅ (Correct)
B) They delete unnecessary websites from the internet. ❌ (Wrong)
C) They randomly store search results with no order. ❌ (Wrong)
D) They display search results in alphabetical order only. ❌ (Wrong)
E) They remove duplicate web pages from the internet. ❌ (Wrong)

Week 12-13: Graph Algorithms

Main Goal:

Learn how **graphs represent relationships**, how they are stored, and how **graph traversal algorithms** work for pathfinding and network analysis.

Lesson 1: Introduction to Graphs

📖 **Concepts to Cover (8 Sections):**

1. What is a Graph?

- A **graph** is a **non-linear data structure** that represents relationships between objects using **nodes (vertices) and edges**.

2. Real-Life Analogy

- **Social media networks**: Users are **nodes**, friendships are **edges**.
- **City maps**: Cities are **nodes**, roads are **edges**.

3. Types of Graphs

- **Undirected Graph**: The connections **do not have direction** (e.g., Facebook friendships).
- **Directed Graph (Digraph)**: The edges **have direction** (e.g., website links, one-way roads).

4. Properties of Graphs

- **Weighted vs. Unweighted**: Edges may have weights (e.g., road distances).
- **Cyclic vs. Acyclic**: Graphs **may or may not have cycles**.

5. Why Use Graphs?

- **Model complex relationships** (e.g., **social media, networks, AI**).
- **Find shortest paths** (e.g., **Google Maps, navigation**).

6. Graph Representation Example (Java)

7. Applications of Graphs

- **Social networks (Facebook, LinkedIn)**.
- **Navigation (Google Maps, Waze)**.
- **AI and Machine Learning (Decision Trees, Neural Networks)**.

8. Conclusion

- Graphs **efficiently represent relationships and real-world problems**.

Activity 1: Matching Type

Match the term with its correct definition:

- **Graph** → A structure with nodes (vertices) connected by edges.
- **Directed Graph** → Edges have a specific direction (arrows).
- **Undirected Graph** → Connections do not have a direction.
- **Node (Vertex)** → A point in the graph.
- **Edge** → A connection between two nodes.

Lesson 2: Graph Representation (Adjacency List & Matrix)

📖 **Concepts to Cover (8 Sections):**

1. How Are Graphs Stored?

- Graphs can be represented using **Adjacency Lists** or **Adjacency Matrices**.

2. What is an Adjacency List?

- Stores **each node with a list of its neighbors**.
- **Efficient for large graphs** (only stores needed connections).

3. What is an Adjacency Matrix?

- A **2D matrix** where a **row-column pair** represents connections.
- **Efficient for dense graphs** (when most nodes are connected).

4. Java Example: Adjacency List vs. Matrix

5. When to Use Each Representation?

- **Adjacency List:** Best for **sparse graphs** (few connections).
- **Adjacency Matrix:** Best for **dense graphs** (many connections).

6. Advantages of Adjacency List

- **Uses less memory for sparse graphs**.
- **Efficient for traversing neighbors**.

7. Advantages of Adjacency Matrix

- **Fast lookup for checking connections**.
- **Easier for implementing algorithms** (e.g., Floyd-Warshall).

8. Conclusion

- The choice between adjacency **list vs. matrix depends on the graph's size**.

Activity 2: Fill in the Blanks

1. **A _____ is a collection of nodes and edges.** (**Answer:** Graph)

2. **If the edges in a graph have direction, it is called a _____.** (**Answer:** Directed Graph)
3. **A node in a graph is also called a _____.** (**Answer:** Vertex)
4. **Social media connections are often represented using a _____.** (**Answer:** Graph)

Lesson 3: Graph Traversal (DFS & BFS)

📖 **Concepts to Cover (8 Sections):**

1. What is Graph Traversal?

- Traversal **visits all nodes in a graph** to find information.

2. Depth-First Search (DFS)

- **Goes deep first** before backtracking.

3. Breadth-First Search (BFS)

- **Explores neighbors first** before moving deeper.

4. DFS vs. BFS (Java Example)

5. When to Use DFS vs. BFS?

- **DFS**: When you want to **explore paths first** (e.g., maze solving).
- **BFS**: When you need to **find the shortest path** (e.g., Google Maps).

6. Real-World Applications

- **DFS:** AI decision trees, backtracking algorithms.
- **BFS:** Shortest path in GPS, social network analysis.

7. Graph-Based AI & Search Engines

- Google **uses graphs** to rank web pages.
- AI **uses decision trees** to find the best moves.

8. Conclusion

- DFS & BFS **help navigate large data structures efficiently**.

Activity 3: Drag and Drop

Match the real-world example to its graph type:

- **Social Media Friendships → Undirected Graph**.
- **Website Links (One page linking to another) → Directed Graph**.
- **Road Map with One-Way Streets → Directed Graph**.
- **Computer Network Connections → Undirected Graph**.

📌 **Week 12-13 Quiz: Graph Algorithms**

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. What is a graph in computer science?

A) A data structure used to represent relationships between objects. ✅ (Correct)
B) A tool used for creating pie charts and bar graphs. ❌ (Wrong)
C) A function that sorts numbers in ascending order. ❌ (Wrong)
D) A special type of database that only stores numbers. ❌ (Wrong)
E) A method for encrypting information in a computer. ❌ (Wrong)

2. How are graphs used in social media?

A) They represent users as nodes and friendships as connections (edges). ✅ (Correct)
B) They help create high-quality images and animations. ❌ (Wrong)
C) They are used to store personal messages in a database. ❌ (Wrong)
D) They prevent users from sending spam messages. ❌ (Wrong)
E) They track only the number of likes on a post. ❌ (Wrong)

3. What's the difference between directed and undirected graphs?

A) A directed graph has arrows indicating direction, while an undirected graph has connections with no specific direction. ✅ (Correct)
B) Directed graphs only store numbers, while undirected graphs store text. ❌ (Wrong)
C) Undirected graphs always have an even number of connections. ❌ (Wrong)
D) Directed graphs always have the same number of edges as nodes. ❌ (Wrong)
E) Undirected graphs are always larger than directed graphs. ❌ (Wrong)

4. What are two ways to store graphs?

A) **Adjacency list and adjacency matrix.** ✅ (Correct)
B) Trees and arrays. ❌ (Wrong)
C) Queues and linked lists. ❌ (Wrong)
D) Binary trees and heaps. ❌ (Wrong)
E) Hash tables and stacks. ❌ (Wrong)

5. What is Depth-First Search (DFS)?

A) A traversal method that explores as far as possible before backtracking. ✅ (Correct)
B) A function used in spreadsheets to format text. ❌ (Wrong)
C) A method for organizing files on a hard drive. ❌ (Wrong)
D) A way to find duplicate files in a folder. ❌ (Wrong)
E) A process for predicting weather patterns. ❌ (Wrong)

6. What is Breadth-First Search (BFS)?

A) A traversal method that explores all neighbors before moving deeper. ✅ (Correct)
B) A system for compressing large files. ❌ (Wrong)
C) A method for predicting the weather using data models. ❌ (Wrong)
D) A process used in photo editing software. ❌ (Wrong)
E) A tool used to design 3D models. ❌ (Wrong)

7. Where is BFS used in real life?

A) Finding the shortest route in navigation apps like Google Maps. ✅ (Correct)
B) Storing contacts in a mobile phone. ❌ (Wrong)
C) Creating a playlist of favorite songs. ❌ (Wrong)
D) Organizing a bookshelf in alphabetical order. ❌ (Wrong)
E) Calculating electricity usage in a household. ❌ (Wrong)

8. Fill in the Blanks

1. **A _____ is a collection of nodes and edges.** (**Answer:** Graph)
2. **If the edges in a graph have direction, it is called a _____.** (**Answer:** Directed Graph)
3. **A node in a graph is also called a _____.** (**Answer:** Vertex)
4. **Social media connections are often represented using a _____.** (**Answer:** Graph)

Week 14-16 – Sorting and Searching

Main Goal:

Understand why **sorting and searching algorithms** are crucial for problem-solving and real-world applications. Learn their implementations, efficiency, and use cases.

Lesson 1: Why Sorting is Important?

📖 **Concepts to Cover (8 Sections):**

1. What is Sorting?

- Sorting is the process of **arranging data in a particular order** (e.g., ascending or descending).

2. Real-Life Analogy: Sorting in Daily Life

- **Libraries:** Books arranged **alphabetically** make searching easy.
- **Supermarkets:** Products are sorted by **categories** for quick access.
- **Online Stores:** Sorting items by **price, relevance, or rating**.

3. Why Do We Need Sorting?

- **Faster Searching** (Binary Search requires sorted data).
- **Efficient Data Processing** (Databases use sorting for indexing).
- **Reduces Computation Time** (Sorting before searching improves speed).

4. Types of Sorting Algorithms

- **Simple Sorting:** Bubble Sort, Selection Sort (Slow, $O(n^2)$).
- **Efficient Sorting:** Quick Sort, Merge Sort (Faster, $O(n \log n)$).

5. Sorting Example (Java - Bubble Sort)

6. Efficiency Comparison

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)

7. Applications of Sorting

- **Search Engines**: Google ranks results using sorting.
- **Finance**: Stock market trends are sorted by date, price, etc.
- **AI & Machine Learning**: Sorting large datasets for training models.

8. Conclusion

Sorting improves **data organization, searching speed, and computational efficiency**.

Activity 1: Matching Type

Match the sorting algorithm with its characteristic:

- **Bubble Sort** → Swaps adjacent elements
- **Quick Sort** → Uses a pivot
- **Merge Sort** → Divides and merges

Lesson 2: Bubble Sort and Selection Sort

📖 **Concepts to Cover (8 Sections):**

1. Bubble Sort - Step by Step

- **Compares adjacent elements** and **swaps if necessary**.
- **Worst case $O(n^2)$, best case O(n) if already sorted**.

2. Selection Sort - Step by Step

- **Finds the smallest element and swaps it to its correct position**.
- **Always takes $O(n^2)$ time** regardless of input.

3. Java Code for Selection Sort

4. Why Are Bubble Sort and Selection Sort Slow?

- **O(n²) complexity** makes them inefficient for large datasets.

5. Applications of Simple Sorting Algorithms

- **Bubble Sort:** Teaching sorting concepts.
- **Selection Sort:** Used when memory swaps are costly.

6. Comparing Bubble Sort & Selection Sort

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Bubble Sort	O(n)	O(n²)	O(n²)
Selection Sort	O(n²)	O(n²)	O(n²)

7. When to Use Simple Sorting?

- **When dataset is small** and sorting speed isn't crucial.

8. Conclusion

Bubble Sort & Selection Sort are easy to implement but slow for large datasets.

Activity 2: Fill in the Blanks

- **_____ repeatedly swaps adjacent elements.** (Bubble Sort)
- **_____ sorting method places the smallest element first.** (Selection Sort)

Lesson 3: Quick Sort and Merge Sort

📖 **Concepts to Cover (8 Sections):**

1. Quick Sort - Divide & Conquer

- **Picks a pivot, sorts smaller elements left & larger right**.
- **Fastest in practice, but worst case is O(n²) if unbalanced**.

2. Merge Sort - Divide & Merge

- **Splits the array, sorts, then merges**.
- **Always O(n log n), but uses more memory**.

3. Java Code for Quick Sort

4. Applications of Fast Sorting

- **Database indexing** (uses Quick Sort).
- **Processing large datasets** (Merge Sort).

5. Quick Sort vs Merge Sort

Algorithm	Best Case	Average Case	Worst Case
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)

6. Why Quick Sort is Used More?

- **Faster in real-world applications** with good pivot selection.

7. Merge Sort's Advantage?

- **Guaranteed O(n log n) performance** but uses extra memory.

8. Conclusion

Quick Sort is used in practice, but Merge Sort is best for large datasets.

Activity 3: Drag and Drop

- **Quick Sort** → Uses Pivot
- **Merge Sort** → Splits and Merges

Week 14-16 Quiz: Sorting and Searching

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. Why do we need sorting?

A) **To organize data so that searching and processing become faster.** ✅ (Correct)
B) To make computers use more memory. ❌ (Wrong)
C) To make data harder to access. ❌ (Wrong)
D) To randomly arrange numbers for testing purposes. ❌ (Wrong)
E) To prevent data from being deleted. ❌ (Wrong)

2. What is Bubble Sort?

A) **A sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order.** ✅ (Correct)
B) A method for finding the largest element in an array. ❌ (Wrong)
C) A sorting technique that directly places elements in their final positions. ❌ (Wrong)

D) A function that compresses data before sorting it. ❌ (Wrong)
E) A process used to convert text into numbers. ❌ (Wrong)

3. What is the difference between Quick Sort and Merge Sort?

A) **Quick Sort picks a pivot and sorts around it, while Merge Sort divides the array into smaller parts and merges them back.** ✅ (Correct)
B) Quick Sort only works on numbers, while Merge Sort only works on text. ❌ (Wrong)
C) Quick Sort always runs faster than Merge Sort in every case. ❌ (Wrong)
D) Merge Sort uses loops, while Quick Sort uses recursion. ❌ (Wrong)
E) Quick Sort sorts in descending order, while Merge Sort sorts in ascending order. ❌ (Wrong)

4. Which sorting algorithm is the fastest in the average case?

A) **Quick Sort** ✅ (Correct)
B) Bubble Sort ❌ (Wrong)
C) Selection Sort ❌ (Wrong)
D) Insertion Sort ❌ (Wrong)
E) None, all are the same ❌ (Wrong)

5. What is Binary Search?

A) **A searching algorithm that repeatedly divides a sorted list in half to find the target value.** ✅ (Correct)
B) A function that converts text into binary code before searching. ❌ (Wrong)
C) A method that searches for an element by randomly guessing its position. ❌ (Wrong)
D) A sorting algorithm that arranges elements before searching. ❌ (Wrong)
E) A search method that works on any unsorted list without modification. ❌ (Wrong)

6. Which one is faster? Why?

A) **Binary Search is faster because it reduces the number of elements checked by half each time (O(log n)), while Linear Search checks every element (O(n)).** ✅ (Correct)
B) Linear Search is faster because it always finds the target in the first step. ❌ (Wrong)
C) Both searches take the same amount of time in all cases. ❌ (Wrong)
D) Binary Search is only faster when searching for numbers but not for words. ❌ (Wrong)
E) Linear Search is always the best option, even for large datasets. ❌ (Wrong)

7. Fill in the Blanks

1. _____ **repeatedly swaps adjacent elements.** (**Answer:** Bubble Sort)
2. _____ **sorting method places the smallest element first.** (**Answer:** Selection Sort)
3. _____ **search requires a sorted array.** (**Answer:** Binary Search)
4. _____ **is faster than Bubble Sort because it has an average time complexity of O(n log n).** (**Answer:** Quick Sort)
5. _____ **splits the list into smaller parts and merges them after sorting.** (**Answer:** Merge Sort)

8. Drag and Drop (Match the Sorting Algorithm to its Key Feature)

- **Bubble Sort → Swaps adjacent elements**
- **Quick Sort → Uses Pivot**
- **Merge Sort → Divides and Merges**
- **Binary Search → Cuts list in half**

Week 17 – Hashing – Efficient Data Storage & Retrieval

Main Goal:

Understand how **hashing optimizes data storage and retrieval** while preventing slow searches. Learn real-world applications, efficiency, and security benefits.

Lesson 1: Introduction to Hashing

📖 **Concepts to Cover (8 Sections):**

1. What is Hashing?

- Hashing is a process that **converts input data into a fixed-length unique code (hash)** using a hash function.
- It is commonly used for **fast data retrieval** in databases, cybersecurity, and caching.

2. Real-Life Analogy: How Hashing Works in Daily Life

- **Library Indexing:** Instead of searching all books, use an index card for fast access.
- **School ID Numbers:** Instead of searching by name, each student has a unique number.
- **URL Shorteners:** Convert long URLs into **short, unique codes** (e.g., bit.ly links).

3. Why is Hashing Useful?

- **Fast Retrieval:** No need to scan all data; **hash function finds it instantly**.
- **Efficient Storage:** Prevents duplicate data and saves memory.
- **Security: Passwords are stored as hashes** so they can't be reversed easily.

4. Hash Functions - How They Work

- A **hash function** takes input and generates a unique hash code.
- Example:

6. Hashing vs. Traditional Searching

Method	**Time Complexity**	**Example**
Linear Search	O(n)	Checking every student's name in a list
Binary Search	O(log n)	Searching in a sorted list

Hashing	O(1)	Directly finding a student's grade using a key

7. Applications of Hashing

- **Databases:** Fast indexing and retrieval.
- **Cybersecurity:** Secure password storage.
- **Blockchain & Cryptography:** Secure transactions.

8. Conclusion

Hashing is used **everywhere** to **speed up searches, store secure data, and reduce memory usage**.

🎯 Activity 1: Matching Type

Match the term with its function:

- **Hash Function** → Converts data into a fixed-size value
- **Hash Table** → Stores key-value pairs
- **Hash Collision** → Occurs when two inputs generate the same hash

---

Lesson 2: How Hash Tables Work & Hash Collisions

📖 Concepts to Cover (8 Sections):

1. What is a Hash Table?

- A **data structure that maps keys to values using a hash function**.
- Example: **Finding a student's grade instantly using their name**.

2. How Hash Tables Store Data

- Instead of scanning an entire list, **hash tables use a key to store and retrieve values instantly**.
- Example:

3. Hash Collisions - Why Do They Happen?

- **A hash collision occurs when two keys generate the same hash value**.
- Example:

4. How to Handle Hash Collisions

- **Chaining:** Store multiple values in the same bucket (linked list).
- **Open Addressing:** Find the next available spot for conflicting data.

5. Efficiency of Hash Tables

Operation	**Average Case**	**Worst Case (with many collisions)**

Insertion	O(1)	O(n)
Lookup	O(1)	O(n)
Deletion	O(1)	O(n)

6. When Not to Use Hash Tables

- **If order is important**, use **arrays or linked lists**.
- **If frequent sorting is needed**, use **trees**.

7. Security Risks of Hashing

- **Hash collisions can be exploited** in cybersecurity attacks.
- **Avoid weak hash functions** (e.g., MD5, SHA1) for sensitive data.

8. Conclusion

Collisions **slow down hashing**, but techniques like chaining and open addressing **prevent slowdowns**.

🎯 **Activity 2: Fill in the Blanks**

1. **_____ is used for password security.** (**Answer:** Hashing)
2. **A _____ stores key-value pairs for quick retrieval.** (**Answer:** Hash Table)
3. **A _____ occurs when two different inputs produce the same hash value.** (**Answer:** Hash Collision)

Lesson 3: Applications of Hashing in Real Life

📖 **Concepts to Cover (8 Sections):**

1. Hashing in Databases

- **Used to quickly index and retrieve records.**
- Example: Student ID → GPA Lookup.

2. Password Hashing in Cybersecurity

- **Stored passwords are hashed instead of saved as plain text.**
- Example Hash:

3. URL Shorteners (e.g., Bit.ly)

- **Long URLs are converted into short, unique hashes**.

4. Blockchain & Cryptography

- **Hashing ensures data integrity** and prevents tampering.

5. Hashing in Load Balancing

- **Used to distribute website traffic efficiently**.

6. Java Code Example: Storing API Keys in a Hash Table

7. Advantages of Hashing in Real Applications

- **Fast lookups, small storage size, better security**.

8. Conclusion

Hashing is used **in almost every modern application**, from security to databases.

🎯 **Activity: Password Hashing Experiment**

- **Show how password hashing works in real-time.**

Week 17 Quiz: Hashing (8 Questions)

📝 **Quiz: Multiple Choice & Fill in the Blanks**

1. What is hashing?

A) **A process that converts data into a fixed-length unique code for fast retrieval.** ✅ (Correct)
B) A method for compressing files to reduce storage size. ❌ (Wrong)
C) A way to organize books in a library using numbers. ❌ (Wrong)
D) A programming language for creating websites. ❌ (Wrong)
E) A function that sorts data in alphabetical order. ❌ (Wrong)

2. Why do computers use hashing?

A) **To store and retrieve data efficiently using unique keys.** ✅ (Correct)
B) To slow down searches and increase security. ❌ (Wrong)
C) To convert numbers into words for easier reading. ❌ (Wrong)
D) To randomly shuffle data for better performance. ❌ (Wrong)
E) To replace databases in all modern applications. ❌ (Wrong)

3. What is a hash table?

A) **A data structure that stores key-value pairs using a hash function for quick access.** ✅ (Correct)
B) A database used to store large amounts of numerical data. ❌ (Wrong)
C) A tool for sorting numbers in ascending order. ❌ (Wrong)
D) A type of memory that permanently saves all data. ❌ (Wrong)
E) A programming language used for AI development. ❌ (Wrong)

4. What is a hash collision?

A) **When two different inputs produce the same hash value.** ✅ (Correct)
B) When a computer crashes while processing a hash function. ❌ (Wrong)
C) When a database fails to store hashed values. ❌ (Wrong)
D) When two computers generate the same password. ❌ (Wrong)
E) When a file is too large to be hashed. ❌ (Wrong)

5. How can we handle hash collisions?

A) **By using techniques like chaining or open addressing.** ✅ (Correct)
B) By deleting duplicate entries in the database. ❌ (Wrong)
C) By restarting the computer to refresh memory. ❌ (Wrong)
D) By using a different programming language. ❌ (Wrong)
E) By avoiding hashing altogether and storing raw data instead. ❌ (Wrong)

6. How is hashing used in databases?

A) **It helps index and retrieve data quickly by mapping keys to values.** ✅ (Correct)
B) It is used to encrypt entire databases for security. ❌ (Wrong)
C) It allows computers to store images and videos more efficiently. ❌ (Wrong)
D) It replaces all traditional sorting methods in databases. ❌ (Wrong)
E) It prevents databases from storing duplicate information. ❌ (Wrong)

7. Fill in the Blanks

1. **_____ is used for password security.** (**Answer:** Hashing)
2. **A _____ stores key-value pairs.** (**Answer:** Hash Table)
3. **A _____ occurs when two different inputs produce the same hash value.** (**Answer:** Hash Collision)
4. **To fix a hash collision, we can use _____ or _____.** (**Answer:** Chaining, Open Addressing)

8. Drag and Drop (Match the Hashing Term to Its Function)

- **Hash Function** → Converts input data into a fixed-size hash value
- **Hash Table** → Stores key-value pairs for quick retrieval
- **Hash Collision** → When two inputs generate the same hash
- **Open Addressing** → A method to handle collisions
