# PBCA303: Data Structure and Algorithms - Unit I Notes

## 1. Introduction to Data Structures

**Definition:**
A data structure is a systematic way of organizing and managing data to efficiently perform operations such as accessing, updating, and storing. It provides a means to manage large amounts of data for efficient processing and retrieval.

**Features of Data Structures:**

1. **Efficient Organization:** Helps in systematic arrangement of data for better efficiency.
2. **Reusability:** Once designed, the structure can be reused across multiple programs.
3. **Performance Enhancement:** Optimizes time and space complexities of algorithms.
4. **Scalability:** Handles growing data effectively.

**Examples:** Arrays, Linked Lists, Stacks, Queues, Trees, Graphs.

## 2. Types of Data Structures

1. **Linear Data Structures:**
   Data elements are arranged sequentially, and each element is connected to its previous and next element.
   - **Examples:** Arrays, Linked Lists, Stacks, Queues.
2. **Non-Linear Data Structures:**
   Data elements are arranged hierarchically, and not in a sequential manner.
   - **Examples:** Trees, Graphs.
3. **Static Data Structures:**
   Fixed size data structures that are defined at compile-time.
   - **Example:** Arrays.
4. **Dynamic Data Structures:**
   Flexible size data structures that can grow or shrink during runtime.
   - **Example:** Linked Lists.

## 3. Algorithm

**Definition:**
An algorithm is a finite sequence of well-defined steps to solve a problem or perform a task.

**Characteristics of a Good Algorithm:**

1. **Finiteness:** Must terminate after a finite number of steps.
2. **Definiteness:** Each step should be clearly defined.
3. **Input:** Accepts zero or more inputs.
4. **Output:** Produces one or more outputs.
5. **Effectiveness:** Steps must be basic enough to be carried out mechanically.

**Example:**
Algorithm to add two numbers:

1. Start
2. Input two numbers `a` and `b`
3. Compute `sum = a + b`
4. Output `sum`
5. Stop

---

## 4. Pseudocode

**Definition:**
Pseudocode is a high-level representation of an algorithm using a combination of natural language and programming constructs. It is language-agnostic and focuses on the logic rather than syntax.

**Example:**
Pseudocode to find the largest of three numbers:

```
START
Input a, b, c
IF a > b AND a > c THEN
    Largest = a
ELSE IF b > c THEN
    Largest = b
ELSE
    Largest = c
ENDIF
Output Largest
STOP
```

---

## 5. Characteristics of Algorithms

1. **Correctness:** Ensures accurate results for all possible inputs.
2. **Efficiency:** Optimal use of resources such as time and memory.
3. **Generality:** Can be applied to a broad range of inputs.
4. **Readability:** Easy to understand and modify.

---

## 6. Algorithm Analysis

Algorithm analysis evaluates the efficiency of an algorithm based on resources like time and space.

### 6.1 Time Complexity

**Definition:**
Measures the amount of time an algorithm takes to complete as a function of the input size.

**Categories:**

1. **Best Case:** Minimum time required.
2. **Worst Case:** Maximum time required.
3. **Average Case:** Expected time under typical conditions.

### 6.2 Space Complexity

**Definition:**
Measures the amount of memory an algorithm needs to execute.

**Components:**

1. **Fixed Part:** Memory required for constants and variables.
2. **Variable Part:** Memory required for dynamic structures.

---

## 7. Abstract Data Types (ADTs)

**Definition:**
An abstract data type is a data model defined by its behavior (operations) rather than its implementation.

**Examples of ADTs:**

1. **List:** Operations include insertion, deletion, traversal.
2. **Stack:** Follows Last In, First Out (LIFO).
3. **Queue:** Follows First In, First Out (FIFO).

---

## 8. String Processing

**Definition:**
String processing involves manipulating strings (sequences of characters) to perform various operations like searching, modifying, or analyzing.

### 8.1 Basic Terminology

1. **String:** Sequence of characters enclosed in quotes.
2. **Substring:** A part of a string.
3. **Length:** Number of characters in a string.

### 8.2 String Operations

1. **Concatenation:** Joining two strings.
   - Example: `"Hello" + "World" = "HelloWorld"`
2. **Substring Extraction:** Extracting part of a string.
   - Example: `Substring("Hello", 1, 3) = "ell"`
3. **Pattern Matching:** Searching for patterns within strings.

### 8.3 Pattern Matching Algorithms

1. **Naive Algorithm:** Compares each character sequentially.
2. **Knuth-Morris-Pratt (KMP) Algorithm:** Uses partial match tables to optimize matching.

# 9. Arrays

**Definition:**
An array is a collection of elements of the same type stored at contiguous memory locations.

### 9.1 Representation in Memory

1. **One-Dimensional Array:** Stored in a single row or column.
2. **Multidimensional Array:** Stored in a matrix-like structure.

### 9.2 Array Operations

1. **Insertion:** Adding an element at a specified index.
2. **Deletion:** Removing an element at a specified index.
3. **Searching:** Finding an element in the array.
   - **Linear Search:** Sequential search through all elements.
   - **Binary Search:** Divides the array into halves for searching (works on sorted arrays).

### 9.3 Sorting Algorithms

1. **Bubble Sort:** Compares adjacent elements and swaps them if necessary.
   **Example:**

```
Input: [5, 2, 9, 1]
Pass 1: [2, 5, 1, 9]
Pass 2: [2, 1, 5, 9]
Pass 3: [1, 2, 5, 9]
```

---

# 10. Multidimensional Arrays

**Definition:**
A multidimensional array is an array of arrays, where data is stored in a tabular form (rows and columns) or in higher dimensions.

### 10.1 Representation:

1. **Two-Dimensional Arrays (Matrices):**
   Represented as rows and columns.
   Example:

```
2D Array:
[ [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9] ]
```

Memory is stored row by row (row-major) or column by column (column-major).

2. **Higher-Dimensional Arrays:**
   For 3D or more dimensions, data is organized in layers.
   Example (3D Array):

```
Layer 1: [ [1, 2], [3, 4] ]
Layer 2: [ [5, 6], [7, 8] ]
```

### 10.2 Advantages of Multidimensional Arrays:

1. Facilitates the representation of data in tabular or matrix form.
2. Useful for complex data modeling (e.g., images, grids).

### 10.3 Applications of Multidimensional Arrays:

1. Graphical representations (images, game boards).
2. Matrices in mathematical computations.
3. Storing and manipulating tabular data.

## 11. Pointers

**Definition:**
A pointer is a variable that stores the memory address of another variable.

### 11.1 Characteristics:

1. Provides dynamic memory allocation.
2. Offers direct memory access.
3. Enables efficient array and string manipulation.

### 11.2 Pointer Example in C:

```c
int a = 10;
int *p = &a; // 'p' stores the address of 'a'
printf("%d", *p); // Output: 10
```

### 11.3 Advantages of Pointers:

1. Enables dynamic memory management.
2. Improves execution speed.
3. Useful for implementing data structures like linked lists.

### 11.4 Disadvantages of Pointers:

1. Can lead to memory leaks if not handled properly.
2. Makes debugging more complex.
3. Prone to errors such as dereferencing null or dangling pointers.

---

## 12. Pointer Arrays

**Definition:**
Pointer arrays are arrays that store the addresses of other variables or arrays.

### 12.1 Example in C:

```c
int a = 10, b = 20, c = 30;
int *arr[3]; // Array of pointers
arr[0] = &a;
arr[1] = &b;
arr[2] = &c;

printf("%d", *arr[1]); // Output: 20
```

### 12.2 Use Cases:

1. Managing collections of data with dynamic memory allocation.
2. Efficient implementation of advanced data structures.

---

## 13. Searching Algorithms

### 13.1 Linear Search

- **Definition:** Sequentially checks each element in the array until the target is found or the end is reached.
- **Complexity:** O(n)

- **Example:**

```
Array: [3, 8, 1, 4]
Search: 1
Steps: 3 → 8 → 1 (Found)
```

### 13.2 Binary Search

- **Definition:** Searches by dividing the array into two halves. Requires the array to be sorted.
- **Complexity:** O(log n)
- **Example:**

```
Array: [1, 3, 5, 7, 9]
Search: 5
Steps: Middle = 5 (Found)
```

---

## 14. Sorting Algorithms

### 14.1 Bubble Sort

- **Definition:** Repeatedly swaps adjacent elements if they are in the wrong order.
- **Complexity:**
  - Best Case: O(n)
  - Worst Case: $O(n^2)$
- **Example:**

```
Array: [4, 3, 2, 1]
Pass 1: [3, 2, 1, 4]
Pass 2: [2, 1, 3, 4]
Pass 3: [1, 2, 3, 4]
```

---