Q1: Define Data Structures and Abstract Data Types.

A: **Data Structures**  
Data Structures (DS) are a way of organizing and storing data in a computer such that it can be retrieved and used effectively. DS considers not only the elements stored but also their relationship to each other. For example, an **array** allows elements to be stored sequentially in memory, making it easy to access them by index. Proper use of data structures makes programs efficient in terms of time and memory.

**Abstract Data Types (ADT)**  
An abstract data type (ADT) is a logical description of the data and the operations that are allowed, independent of the actual implementation. For example, integers form an ADT: they are defined by values like 0, 1, -1, 2, … and operations such as addition, subtraction, multiplication, etc., regardless of how they are stored in memory. Similarly, a **list** can be implemented using arrays or linked lists, but the user is abstracted from this detail.

**Q2. Explain Linear and Non-Linear Data Structure with example.**

**A: Linear Data Structures**  
In linear data structures, elements are organized in a sequential manner, one after another. Each element is directly connected to the next, and only one element can be reached from another during traversal. Examples include **arrays, stacks, queues, and linked lists**. For instance, in a queue, elements are inserted at the rear and deleted from the front, which maintains a strict linear order.

**Non-Linear Data Structures**  
In non-linear data structures, elements are not arranged sequentially. Instead, one data element can be connected to multiple elements, forming hierarchical or network-like structures. Examples include **trees and graphs**. A tree represents parent-child relationships, while graphs can represent real-world networks such as social networks or road maps.

**Q3. Explain the classification diagram representing various types of data structures.**

A: Data structures are broadly classified into two categories:

* **Primitive Data Structures:** These are the basic structures provided by the programming language and operated directly by machine instructions. Examples include integers, floating-point numbers, characters, and pointers.
* **Non-Primitive Data Structures:** These are derived from primitive data structures and emphasize structuring of homogeneous or heterogeneous elements. Examples include lists, stacks, queues, trees, and graphs.
  + Non-primitive structures are further divided into:
    - **Linear structures:** e.g., array, stack, queue, linked list.
    - **Non-linear structures:** e.g., tree, graph.

This classification ensures we can choose the correct structure for solving specific problems efficiently.

**Q4. Explain Static and Dynamic Data Structures with example.**

**A: Static Data Structures**  
Static data structures have a fixed memory size determined at compile time. Once declared, their size cannot change during execution. For example, an **array** of size 10 always holds 10 elements, no more, no less. These structures are easy to implement and access since memory allocation is simple and sequential. However, they may waste memory if the allocated size is larger than needed.

**Dynamic Data Structures**  
Dynamic data structures allow memory size to be changed at runtime. For example, a **linked list** grows or shrinks as nodes are inserted or deleted. This flexibility makes them suitable for applications where the amount of data is not known in advance. However, they are more complex to implement because of pointers and dynamic memory allocation.

**Q5. Explain the necessary characteristics of an algorithm.**

A: An algorithm must satisfy the following characteristics:

* **Non-Ambiguity:** Each step in an algorithm should be clear and precise, leaving no room for misinterpretation.
* **Finiteness:** The algorithm should terminate after a finite number of steps. Infinite loops make an algorithm invalid.
* **Speed (Efficiency):** An algorithm should execute efficiently, producing the desired result quickly. Efficiency is often measured in terms of time and memory usage.
* **Correctness:** The algorithm must always produce the correct output for valid input.

Together, these characteristics ensure the algorithm is practical, reliable, and efficient for solving problems.

**Q6. Describe the Asymptotic Notations Ω, θ and O.**

A:

* **O(g(n)) – Big Oh Notation (Upper Bound):** Represents the maximum growth rate of an algorithm. It gives an upper limit on time complexity. Example: 10n ∈ O(n²) means the function does not grow faster than n².
* **Ω(g(n)) – Big Omega Notation (Lower Bound):** Represents the minimum growth rate. It describes the best-case scenario of an algorithm. Example: 10n² ∈ Ω(n²) shows the function grows at least as fast as n².
* **Θ(g(n)) – Big Theta Notation (Tight Bound):** Represents the exact growth rate when both lower and upper bounds are the same. Example: n(n+1)/2 ∈ Θ(n²).

These notations help in analyzing and comparing algorithms independent of hardware or implementation.

**Q7. Compare static and dynamic data structures.**

**A: Static Data Structures**

* Have a fixed memory size decided at compile time.
* Example: **Arrays**.
* Easy to implement and provide fast access using indices.
* Limitation: May waste memory or run out of space if the declared size is too small.

**Dynamic Data Structures**

* Size can change during runtime based on operations.
* Example: **Linked List**.
* Provide flexibility and efficient use of memory.
* Limitation: Implementation is more complex due to dynamic memory allocation and pointers.

Thus, static structures are suitable when data size is known, while dynamic structures are preferred when data size is unpredictable.

**Q8. Compare Linear and Non-linear data structures.**

**A: Linear Data Structures**

* Organize data in a sequential manner.
* Each element is connected to the next, so traversal is one-by-one.
* Examples: **Array, Stack, Queue, Linked List**.
* Suitable for simple data storage and operations like searching and sorting.

**Non-Linear Data Structures**

* Organize data in a hierarchical or networked manner.
* Each element can be connected to multiple elements.
* Examples: **Tree, Graph**.
* Useful for representing hierarchical relationships (like file systems) or networks (like maps, social networks).