1. Define Data Structures and Abstract Data Types.

Data Structures:
A *data structure* is a systematic way of organizing, managing, and storing data in a computer so that it can be accessed and modified efficiently. Examples include arrays, linked lists, stacks, queues, trees, and graphs.

Abstract Data Types (ADTs):
An *abstract data type* is a logical description of how data is viewed and the operations allowed on it, without specifying how these operations will be implemented. Examples are Stack, Queue, List, and Deque.

2. Explain Linear and Non-Linear Data Structure with example.

Linear Data Structure:
A linear data structure is one in which data elements are arranged sequentially, and each element is connected to its previous and next element.

- Examples: **Array, Linked List, Stack, Queue**

Non-Linear Data Structure:
A non-linear data structure is one in which data elements are not arranged in a sequence. Instead, they are arranged hierarchically or in a network structure, where one element can be connected to multiple elements.

- Examples: **Tree, Graph**

3. Explain the classification diagram representing various types of data structures.

Classification of Data Structures:

Data structures are broadly classified into two categories:

1. **Primitive Data Structures** – These are the basic structures supported directly by the programming language.
    - Examples: Integer, Float, Character, Boolean.
2. **Non-Primitive Data Structures** – These are more complex and are derived from primitive data types. They are further classified into:
    - **Linear Data Structures:** Elements are arranged in a sequential manner.
        - Examples: Array, Linked List, Stack, Queue.
    - **Non-Linear Data Structures:** Elements are arranged hierarchically or graph-like.
        - Examples: Trees, Graphs.

4. Explain Static and Dynamic Data Structures with example.

Static Data Structure:

- A static data structure has a fixed size, decided at compile time.
- Memory is allocated once and cannot be changed during program execution.
- Example: **Array** → int arr[10]; (size fixed as 10).

Dynamic Data Structure:

- A dynamic data structure can change size during program execution.
- Memory is allocated and deallocated as needed (using pointers).
- Example: **Linked List** → nodes can be created or deleted at runtime.

5. Explain the necessary characteristics of an algorithm.

Necessary Characteristics of an Algorithm:

1. **Finiteness** – An algorithm must always terminate after a finite number of steps.
2. **Definiteness** – Each step must be clear, well-defined, and unambiguous.
3. **Input** – An algorithm should have zero or more inputs.
4. **Output** – An algorithm must produce at least one output (result).
5. **Effectiveness** – Each step must be simple, basic, and can be performed in a finite time.
6. **Generality** – An algorithm should be applicable to a broad set of problems, not just one specific case.

6.Describe the Asymptotic Notations Ω, θ and O.

Asymptotic Notations are used to describe the efficiency of an algorithm in terms of time or space complexity as the input size grows.

1. **Big-O Notation (O):**
    - Describes the **upper bound** of an algorithm's growth rate.
    - Represents the *worst-case* performance.
    - Example: If an algorithm takes at most $c \cdot n^2$ steps, it is $O(n^2)$.
2. **Theta Notation (θ):**
    - Describes the **tight bound** (average case).
    - Represents both *upper and lower* bounds.
    - Example: If an algorithm always takes $c_1 \cdot n^2$ to $c_2 \cdot n^2$ steps, it is $\theta(n^2)$.
3. **Omega Notation (Ω):**
    - Describes the **lower bound** of an algorithm's growth rate.
    - Represents the *best-case* performance.
    - Example: If an algorithm takes at least $c \cdot n^2$ steps, it is $\Omega(n^2)$.

7. Compare static and dynamic data structures.

Feature	**Static Data Structure**	**Dynamic Data Structure**
Memory Allocation	Fixed, decided at compile time	Flexible, allocated at runtime

Feature	Static Data Structure	Dynamic Data Structure
Size	Cannot be changed during execution	Can grow or shrink during execution
Implementation	Simple and easy (e.g., Arrays)	Complex (uses pointers, e.g., Linked List)
Memory Utilization	May waste memory if not fully used	Efficient memory use, allocated as required
Speed	Faster access (direct indexing)	Slower (extra pointer management)
Examples	Array, Static Stack	Linked List, Dynamic Queue, Tree

8. Compare Linear and Non-linear data structures.

Feature	Linear Data Structure	Non-Linear Data Structure
Arrangement	Elements stored sequentially (one after another).	Elements stored hierarchically or network-like.
Traversal	Traversed in a single run (one by one).	May require multiple runs

Feature	Linear Data Structure	Non-Linear Data Structure
		(parent–child, edges).
Memory Utilization	Memory may be wasted if size is fixed (e.g., arrays).	Memory is used efficiently using pointers.
Complexity	Simple to implement and use.	More complex to implement and manage.
Examples	Array, Linked List, Stack, Queue.	Tree, Graph.