Introduction and Overview

Chapter 01

BASIC TERMINOLOGY

Data: Values or sets of values.

Data Item: A single unit of value.

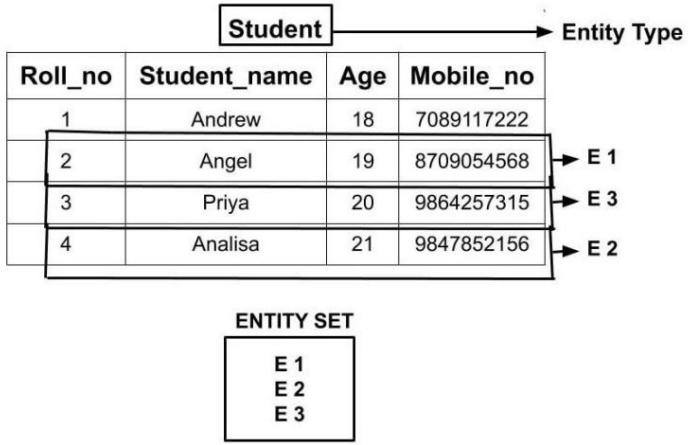
- **Elementary Item**: Cannot be divided further.
  ➢ *e.g., Social Security Number*

- **Group Item**: Composed of subitems.
  ➢ *e.g., Name → First Name, Middle Initial, Last Name*

Information: The term "information" is sometimes used for data with given attributes, or, in other words, meaningful or processed data.

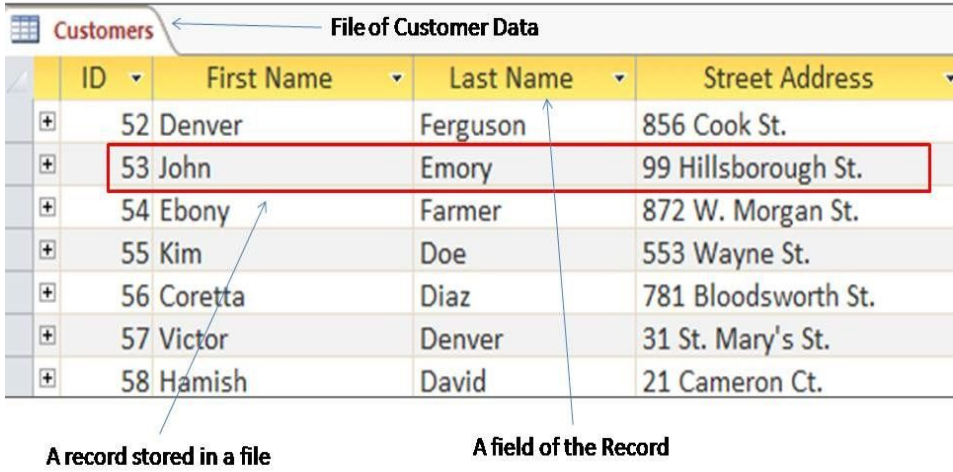
BASIC TERMINOLOGY

- **Entity**: An entity is something that has certain attributes or properties which may be assigned values.
  ➢ Example: Employee
- **Entity Set**: Entities with similar attributes form an entity set.
  ➢ All employees in an organization
- **Attributes**: Properties of an entity.
  ➢ Name, Age, Gender, Social Security Number
- **Values:** Specific data for attributes.
  ➢ Priya , 34, F,13423



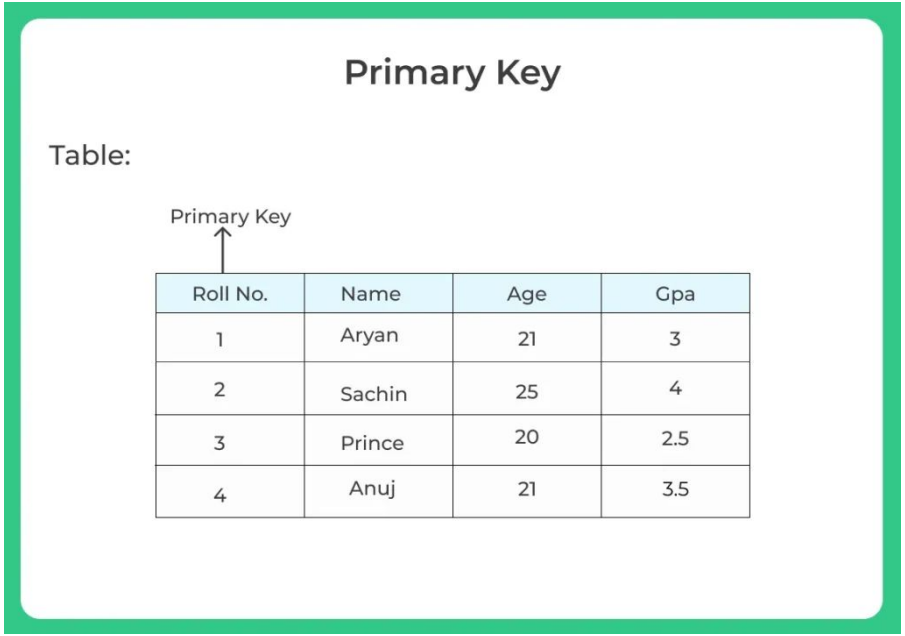
BASIC TERMINOLOGY

- **Field:** A single elementary unit of information representing an attribute of an entity.
  - ➢ Name = Priya
- **Record:** The collection of field values of a given entity.
  - ➢ [Name, Age, Gender, SSN]
- **File:** The collection of records of the entities in a given entity set.
  - ➢ Employee records file



Primary Key

- **A field K where its values k_1, k_2,... uniquely determine a record is called a primary key.**
- **The values in such a field are called keys or key values.**



Record Types

- **Fixed-Length:** All records have same size.

  ➢ E.g., Vehicle records

- **Variable-Length:** Different record sizes.

  ➢ E.g., Student records with varying course counts

Subject Matter of Data Structures

The study of data structures typically involves the following **three key steps**:

1. Logical or mathematical description of the structure
2. Implementation of the structure on a computer
3. Quantitative analysis of the structure, which includes determining the amount of memory needed to store the structure and the time required to process the structure.

Data Structure Definition

Data may be organized in many different ways; the logical or mathematical model of a particular organization of data is called a data structure.

The choice of a particular data model depends on two considerations:

- It must be rich enough in structure to mirror the actual relationships of the data in the real world.
- It should be simple enough that one can effectively process the data when necessary.

Classification of Data Structures

Data structures are generally classified into **primitive** and **non-primitive** data structures.

Primitive Data Structures

- Basic data types such as **integer**, **real**, **character**, and **boolean**.
- These data types consist of characters that **cannot be divided**, and hence they are also called **simple data types**.

Non-Primitive Data Structures

- Example: **Processing of complex numbers** (very few computers support this directly).
- Other examples include:
  Linked Lists, **Stacks**, **Queues**, **Trees**, and **Graphs**.

Classification of Data Structures

Non-Primitive Data Structures are classified into **linear and non-linear** data structures.

Linear Data Structures

- A data structure is said to be **linear** if its elements form a **sequence or a linear list**.
- The data is arranged in a **linear fashion**, although memory storage **need not be sequential**.
- Examples: **Arrays**, **Linked Lists**, **Stacks**, **Queues**

Non-Linear Data Structures

- A data structure is said to be **non-linear** if the data is **not arranged in sequence**.
- **Insertion and deletion** are not possible in a strictly linear manner.
- Examples: **Trees**, **Graphs**

Classification of Data Structures

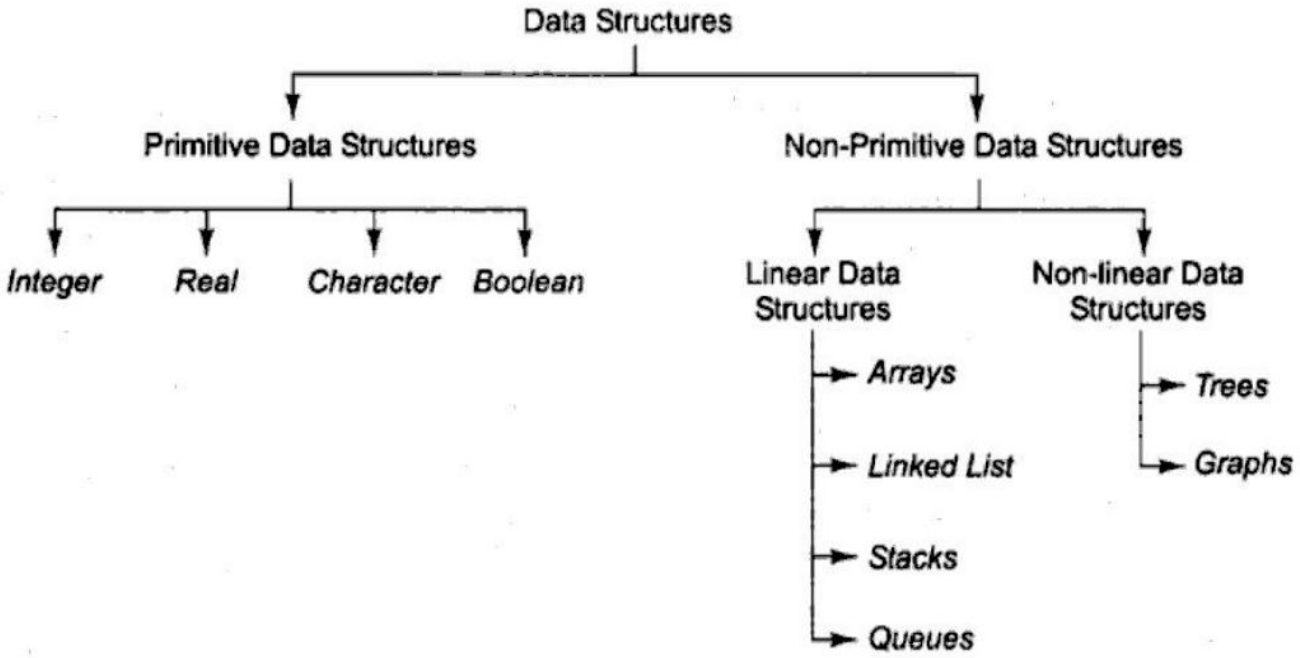


Fig. 1.1 Classification of Data Structures

Primitive Data Structure

Integer: The integer data type contains the numeric values. It contains the whole numbers that can be either negative or positive. When the range of integer data type is not large enough then in that case, we can use long.

Float: The float is a data type that can hold decimal values. When the precision of decimal value increases then the Double data type is used.

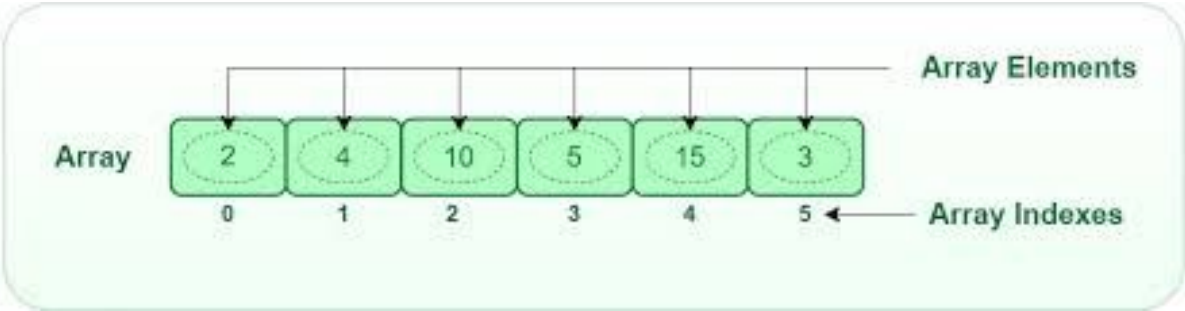
Boolean: It is a data type that can hold either a True or a False value. It is mainly used for checking the condition.

Character: It is a data type that can hold a single character value both uppercase and lowercase such as 'A' or 'a'.

Non-Primitive Data Structure (Linear)

Arrays : A collection of **similar data elements** stored in **contiguous memory locations**.

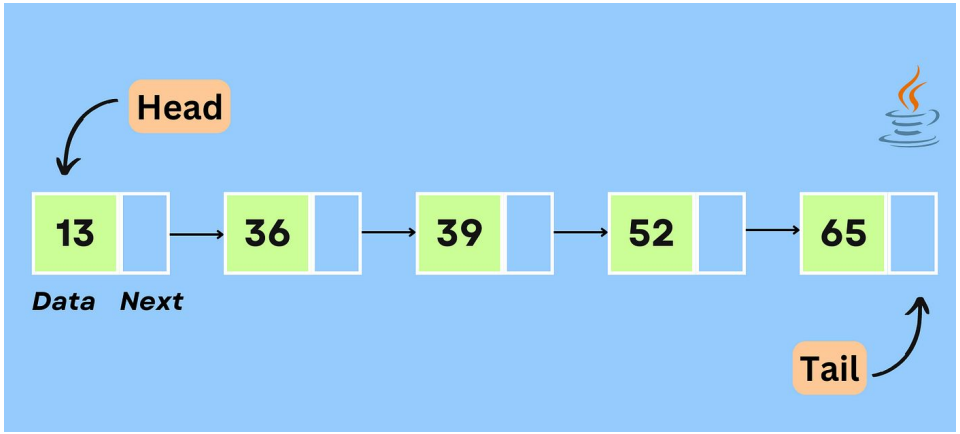
- One-dimensional array: A[1], A[2], ..., A[N]
- Two-dimensional array: Result[student, semester], e.g., Result[Himel,6th] = 3.54



Non-Primitive Data Structure (Linear)

Linked Lists : A collection of elements (nodes), where each node points to the next using a link or pointer.

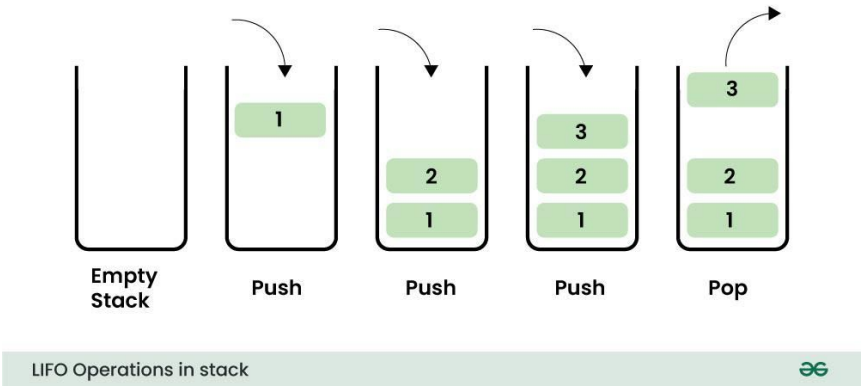
- A salesperson pointing to a list of their customers using pointers.



Non-Primitive Data Structure (Linear)

Stacks : A Last-In-First-Out (LIFO) data structure. Insertion and deletion happen at the top only.

- Like a stack of plates—you can only add or remove the top plate.



LIFO Operations in stack

Non-Primitive Data Structure (Linear)

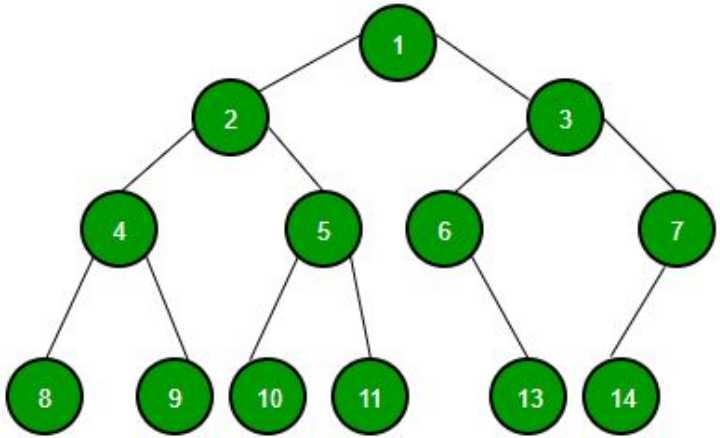
Queues : A First-In-First-Out (FIFO) data structure. Insertion happens at the tail, deletion happens from the front.

- People waiting in a queue at a bus stop.

Queue Data Structure

Non-Primitive Data Structure (Non-Linear)

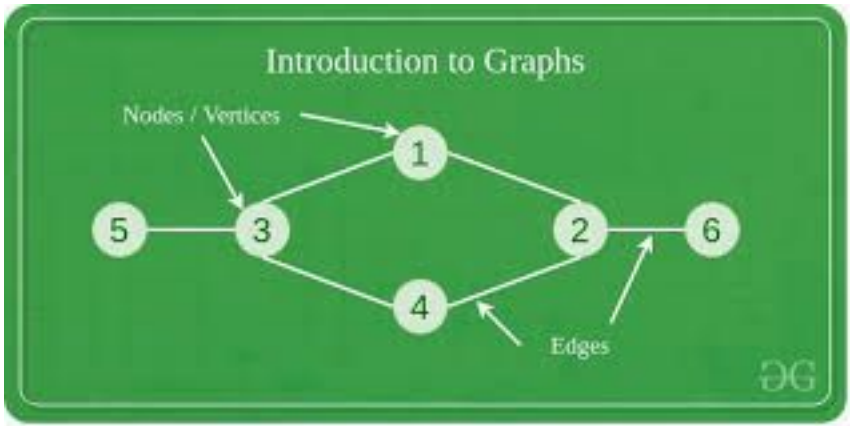
Trees : A hierarchical, non-linear data structure with a root node and child nodes.



Non-Primitive Data Structure (Non-Linear)

Graphs : A non-linear data structure showing relationships between pairs of elements, not necessarily hierarchical.

- Airline routes between cities: A graph showing connections like Boston ↔ New York ↔ Chicago.



Linear Vs Non Linear Data Structure

Linear Data Structure	Non-Linear Data Structure
Every item is related to its previous and next time.	Every item is attached with many other items.
Data is arranged in linear sequence.	Data is not arranged in sequence.
Data items can be traversed in a single run.	Data cannot be traversed in a single run.
Eg. Array, Stacks, linked list, queue.	Eg. tree, graph.
Implementation is easy.	Implementation is difficult.

Data Vs Data Structure

Data	Data Structure
Raw facts, figures, or values without any context or organization.	A way of organizing, managing, and storing data so it can be accessed efficiently.
Unprocessed or unorganized information.	Logical or mathematical model to represent structured data.
Stored in simple variables or constants.	Stored using memory structures like nodes, pointers, and indexes.
Cannot be processed directly for complex operations.	Enables efficient data manipulation and retrieval.
`"John"`, `23`, `3.14`, `'A'`, `True`	Arrays, Linked Lists, Stacks, Queues, Trees, Graphs

Operations of Data Structure

The following **four** operations play a major role:

1. Traversing: Accessing each record exactly once so that certain items in the record may be processed. (This accessing and processing is sometimes called "visiting" the record.)

2. Searching: Finding the location of the record with a given key value, or finding the locations of all records which satisfy one or more conditions.

3. Inserting: Adding a new record to the structure.

4. Deleting: Removing a record from the structure.

Applications of Data Structure

- Used in databases, file systems, and memory management to store and access data quickly.
- Many algorithms rely on data structures like stacks, queues, trees, and graphs to solve problems efficiently (e.g., DFS/BFS in graphs).
- Symbol tables, syntax trees, and parsing techniques in compilers are implemented using data structures.
- Graphs are used to represent and optimize routes in networks like the internet or GPS systems.
- Trees (e.g., decision trees), graphs, and hash tables are used for logic building and quick access to learned data.
- Scheduling, memory allocation, and process management in operating systems use queues, heaps, and linked lists.

What is an Algorithm?

Definition: An **algorithm** is a well-defined list of steps or instructions designed to solve a particular problem.

Efficiency Measures: The two major measures of an algorithm's efficiency are:

- **Time Complexity:** How much time an algorithm takes to run, depending on the size of the input.
- **Space Complexity:** How much memory or space an algorithm requires during execution.

Complexity:
The **complexity** of an algorithm is a function that relates the running time and/or space required to the size of the input. (This will be studied further in Chapter 2.)

Characteristics of an Algorithm

Well-defined Steps: Each step of the algorithm must be clear and unambiguous.

Input: An algorithm should have zero or more inputs, i.e., the data it works on.

Output: The algorithm produces at least one output, i.e., the result of processing the input.

Finiteness: The algorithm must always terminate after a finite number of steps.

Effectiveness: Every step must be basic enough to be carried out, in principle, by a person or machine using a finite amount of resources.

Deterministic: The algorithm should produce the same output for the same input every time it runs.

Searching Algorithms

Searching algorithms are used to find a specific item (such as a name, ID, or value) in a data structure like an array, list, or file.

Example Scenario:

Suppose we have a **membership file** where each record has:

- Member's Name
- Telephone Number

We are given a **member's name**, and we need to find their **telephone number**.

Searching Techniques (Linear Search)

Search each record one by one until the target name is found.

Algorithm:

- Start from the first record.
- Compare each name with the target name.
- Stop when a match is found.

Time Complexity:

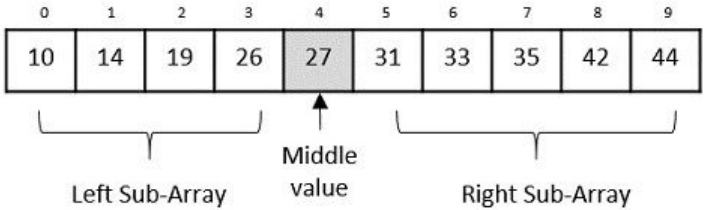
- Average case: C(n) = n/2
- Worst case: C(n) = n

Searching Techniques (Binary Search)

Efficient search method used on **sorted** lists. It repeatedly divides the search interval in half.

Algorithm:

1. Compare the target name with the **middle** item.
2. If equal, return it.
3. If less, search the **left half**.
4. If greater, search the **right half**.
5. Repeat until found.



Time Complexity:

- Best case: 1 comparison
- Worst case: $\log_2(n)$ comparisons

What is a Time-Space Tradeoff?

A time-space tradeoff occurs when you use more memory (space) to make a program faster (less time), or you reduce memory usage at the cost of slower execution.

In simple terms:

- Use more space to save time
- Use more time to save space

Time-Space Tradeoff Example

Example: Searching in a File

Suppose you have a file containing:

- Names
- Social Security Numbers (SSNs)
- Other data

If the file is **sorted by name**, you can do a **binary search** to find a name quickly.
But if you need to search by **SSN**, you would have to use **linear search**, which is slow.

The End!