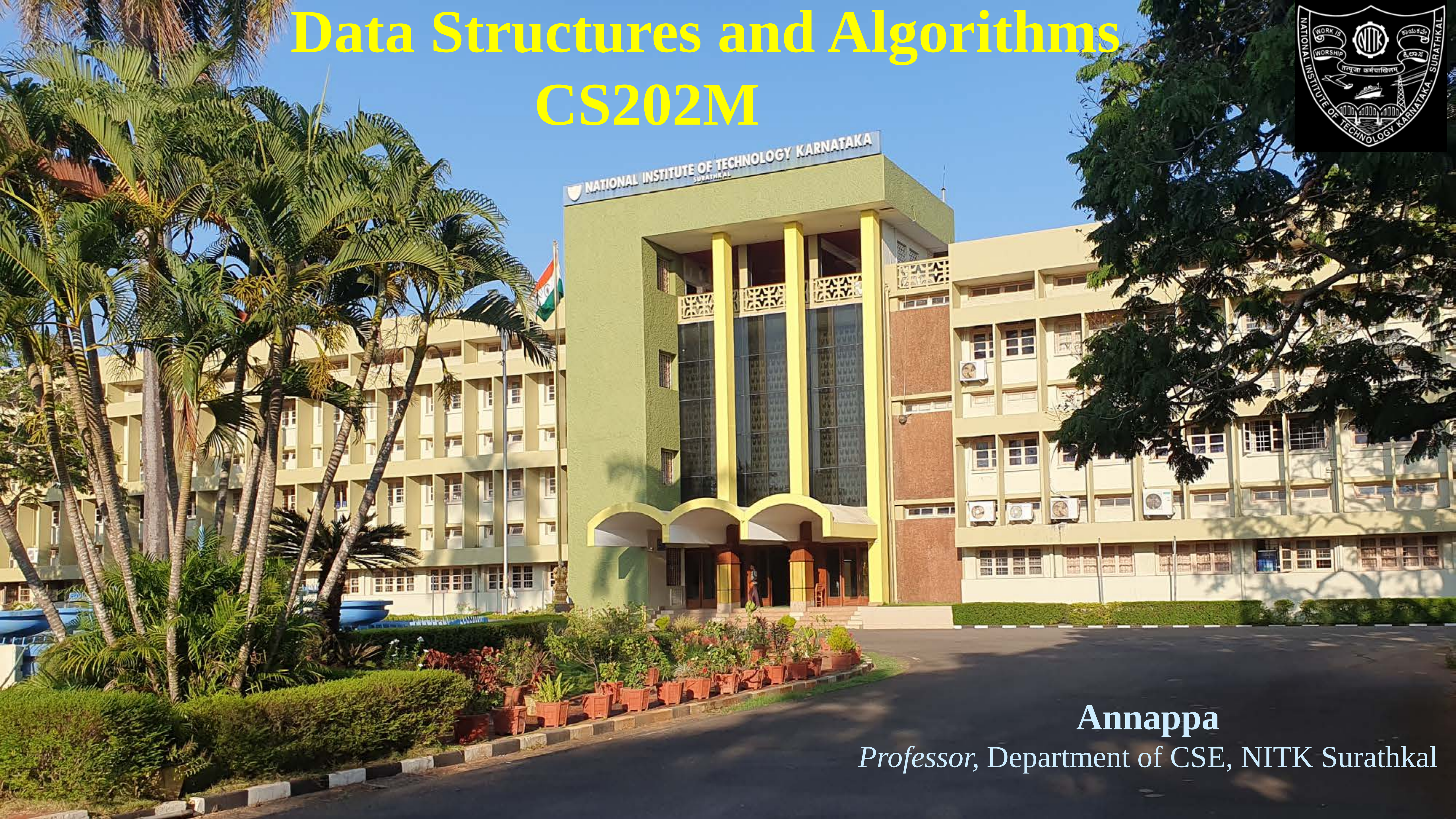
Data Structures and Algorithms
CS202M



Annappa
Professor, Department of CSE, NITK Surathkal

INTRODUCTION
TO
DATA STRUCTURES

Good Programs

There are a number of facets to good programs:

Program/s must

- **run correctly**
- **run efficiently**
- **be easy to read and understand**
- **be easy to debug and**
- **be easy to modify.**

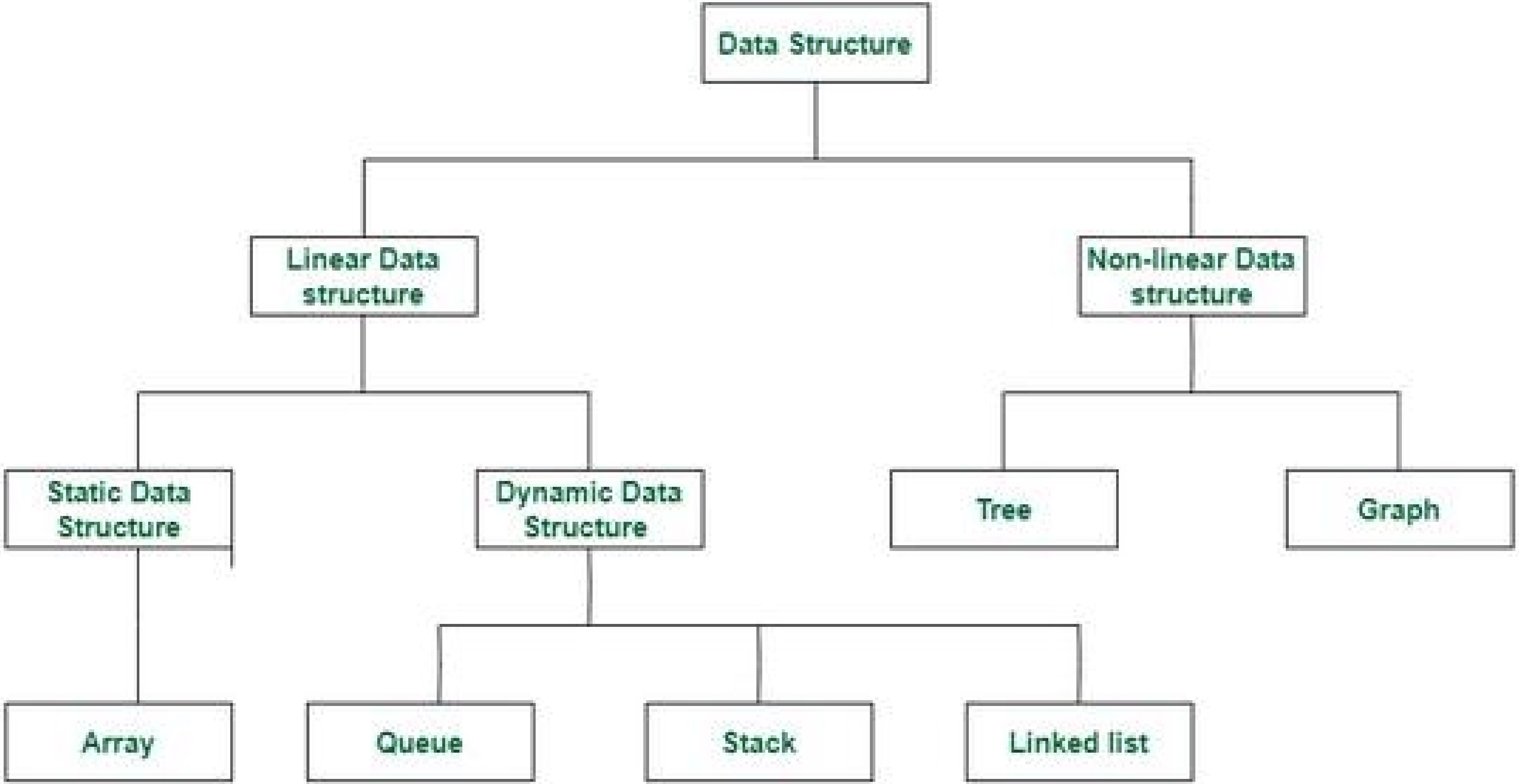
Data Structures - Introduction

- Most fundamental and built-in concept in Computer Science and Engineering.
- Good knowledge of Data Structure is a **must** to design and develop **efficient software systems**.
- A data structure is a way to **store and organize data** in computer, so that it can be **accessed** and **used efficiently**.

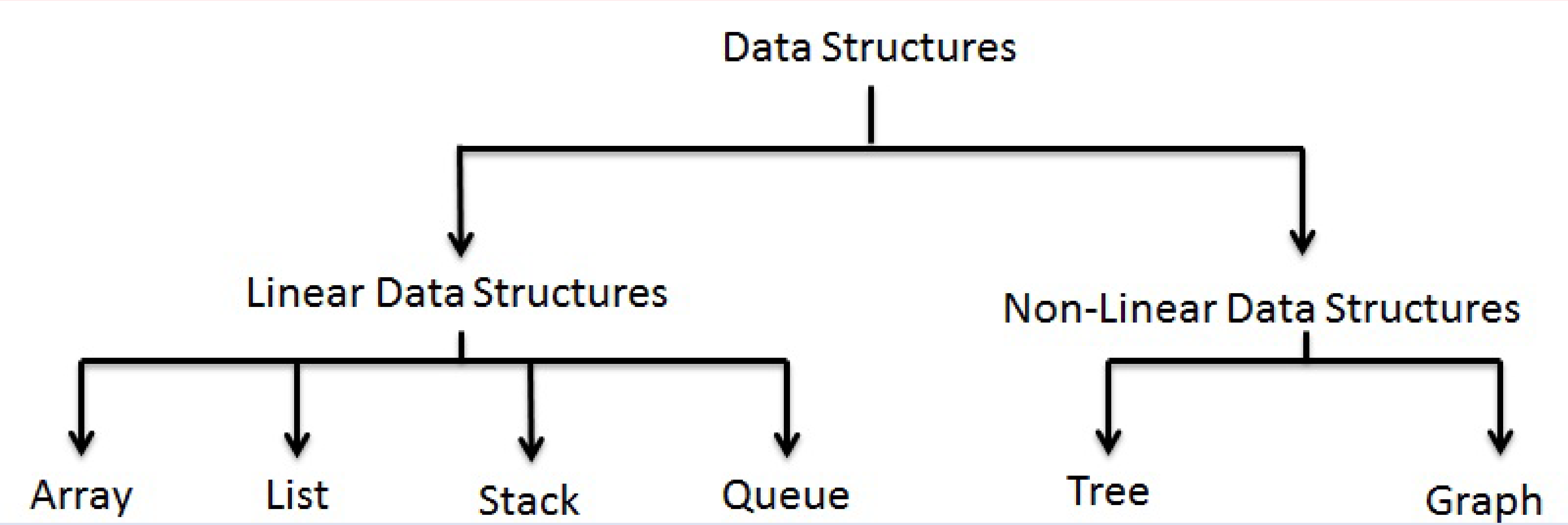
Data Structures - Introduction

- The choice of data structure can make the difference between a program running in a few seconds or requiring many days.
- A data structure requires a certain amount of **space** for each data item it stores, a certain amount of **time** to perform a single basic operation, and a certain amount of **programming effort**.

Classification of Data Structure



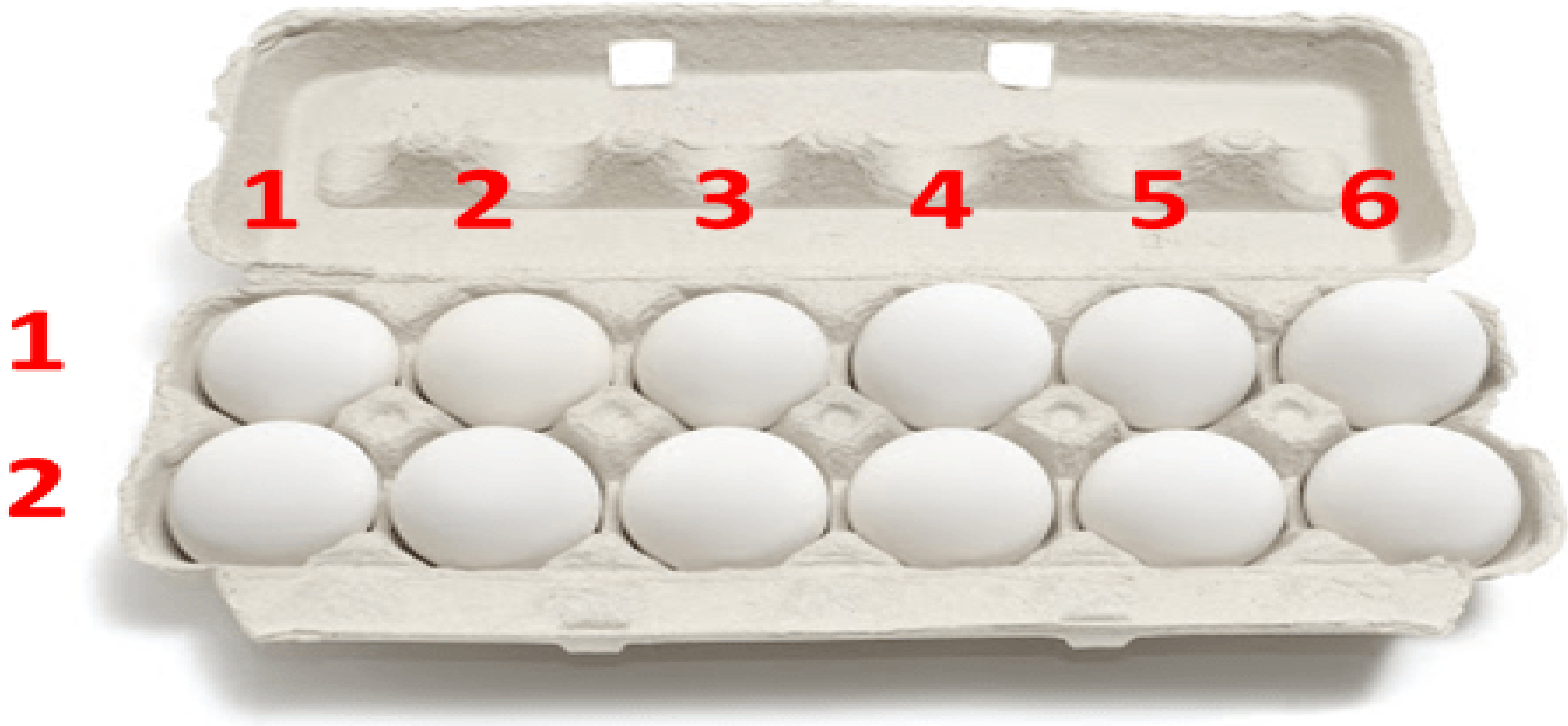
Classification of Data Structure



When studying Data Structure the following things are important

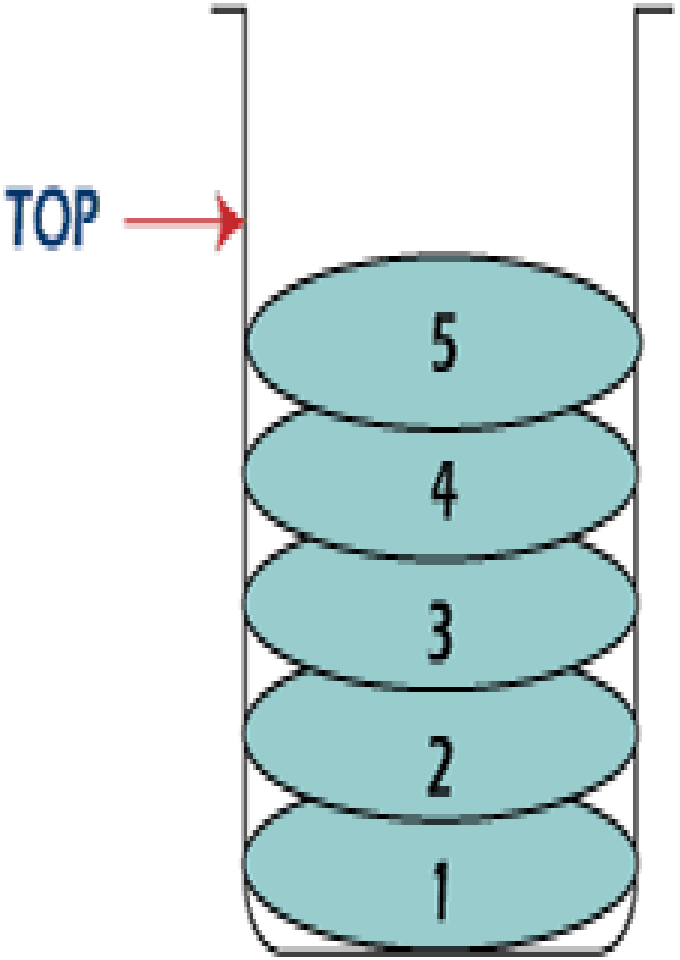
1. Logical view
2. Operations
3. Cost of operations
4. Implementation

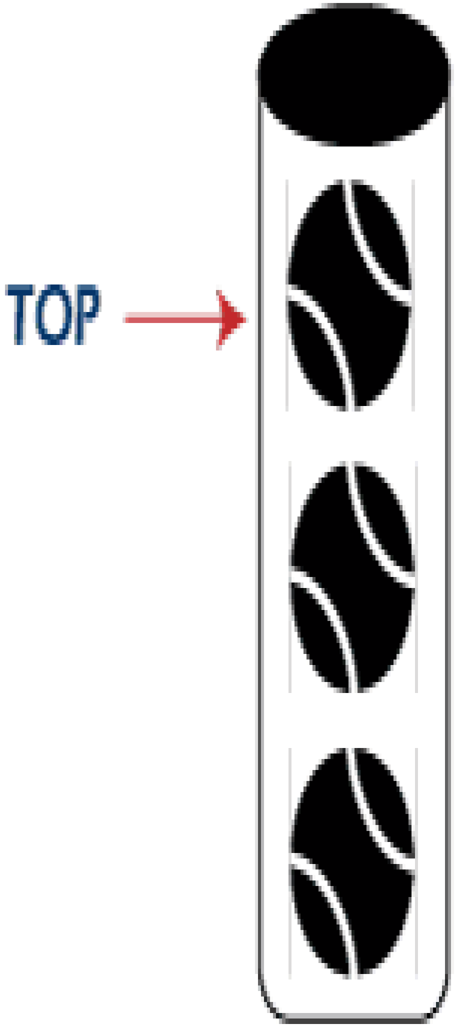
Which Data Structure is this?

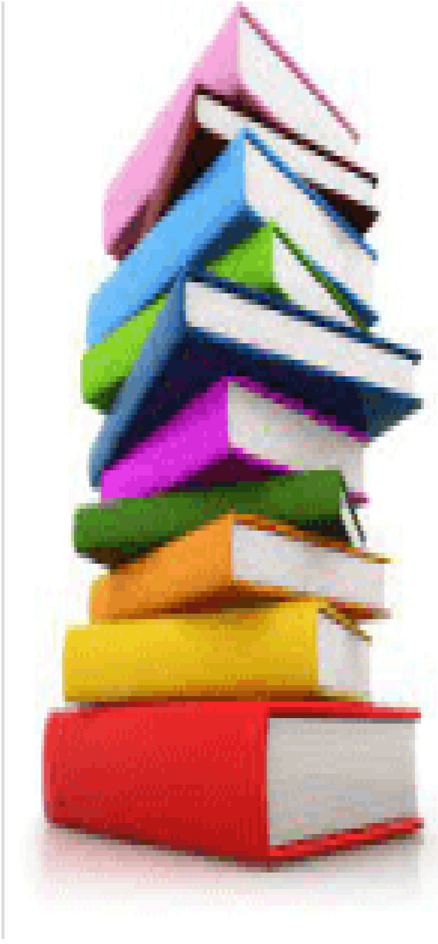


ARRAY

Which Data Structure is this?







STACK

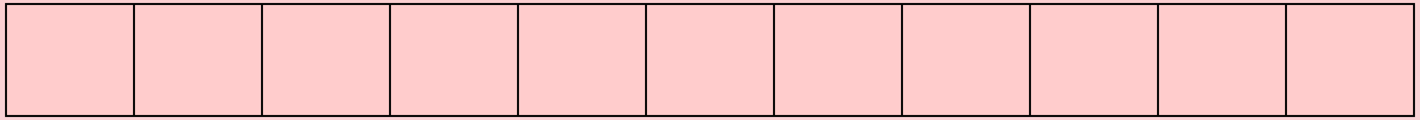
Which Data Structure is this?

QUEUE

Which Data Structure is this?

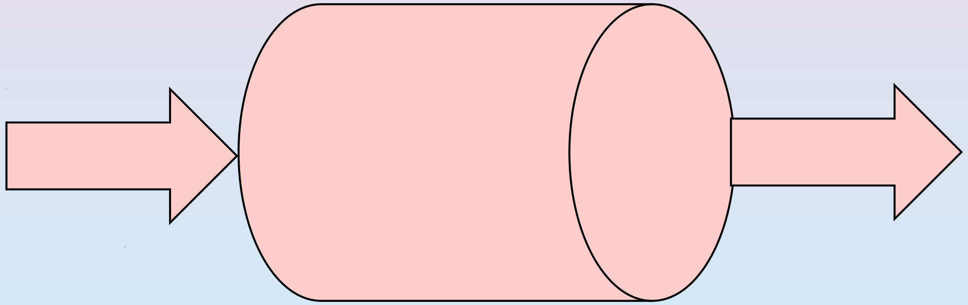
Linked list

Linear Data Structures

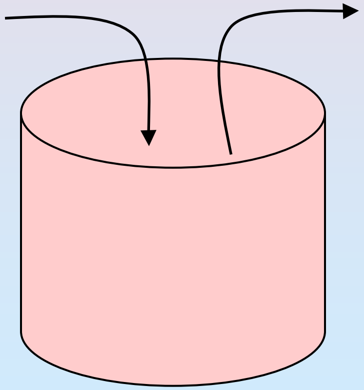


Array

Linked list

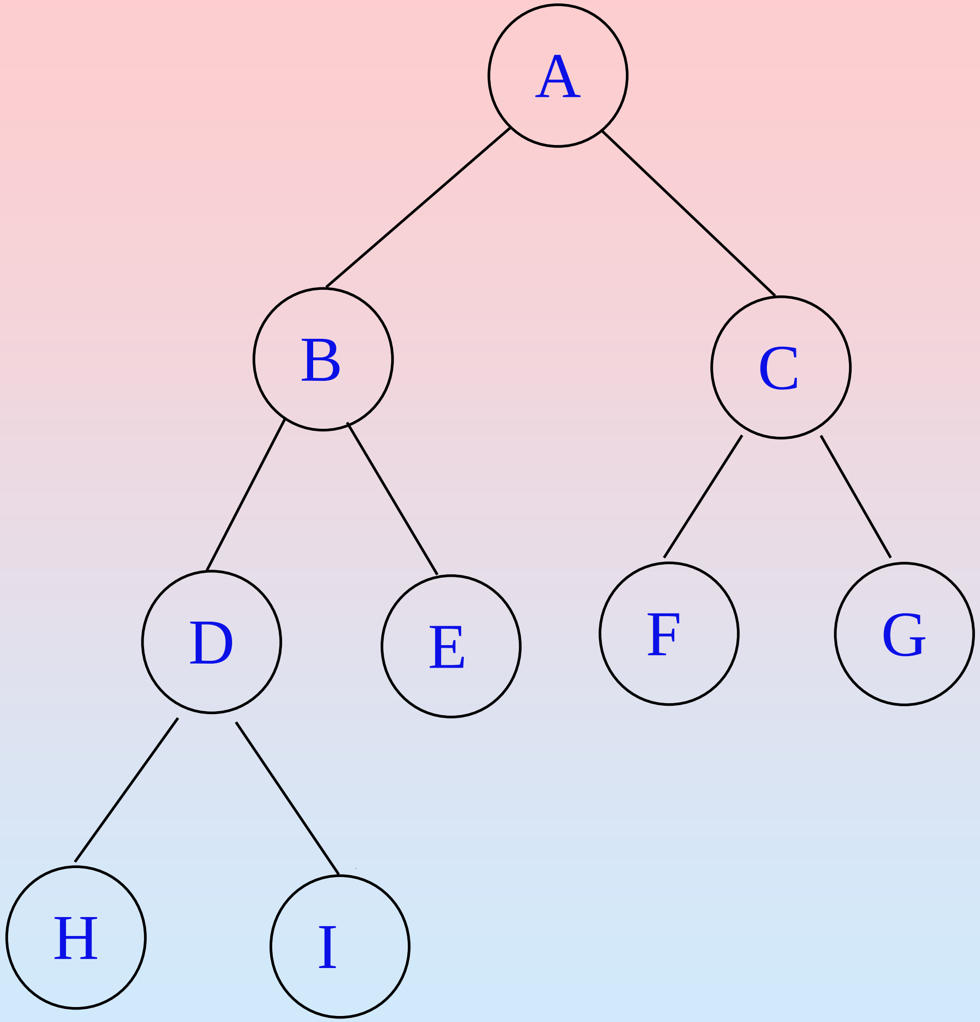


Queue

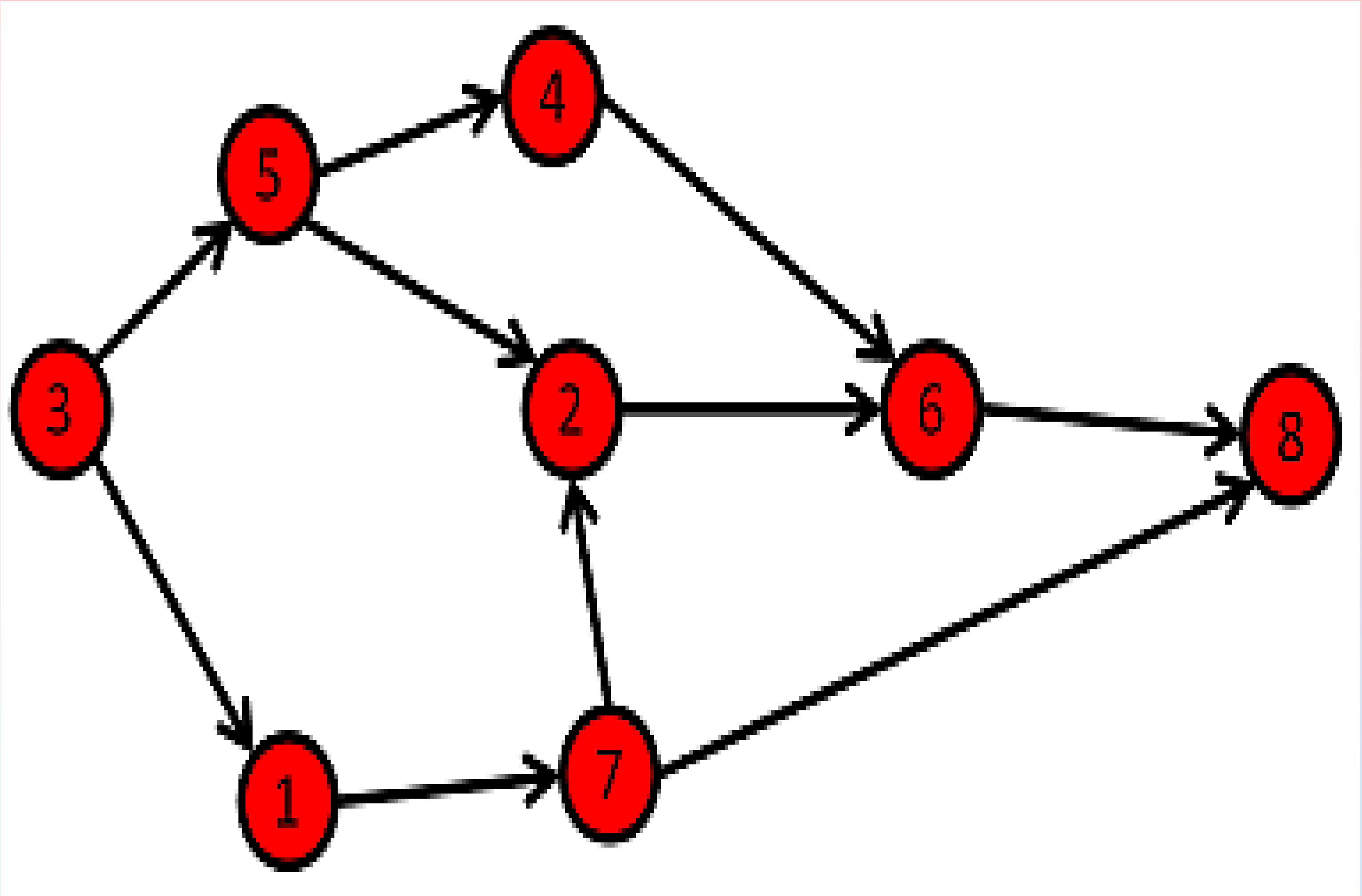


Stack

Non-Linear Data Structures



Tree



Graph

Linear Data Structures vs.	Non Linear Data Structures
The data items are arranged in sequential order, one after the other.	The data items are arranged in non-sequential order (hierarchical manner).
All the items are present on the single layer.	The data items are present at different layers.
It can be traversed on a single run. That is, if we start from the first element, we can traverse all the elements sequentially in a single pass.	It requires multiple runs. That is, if we start from the first element it might not be possible to traverse all the elements in a single pass.
The memory utilization is not efficient.	Different structures utilize memory in different efficient ways depending on the need.
Are relatively easier to implement.	Requires a higher level of understanding and are more complex to implement.
Time complexity increases with the increase in the input size.	Time complexity of non-linear data structure often remains same with the increase in the input size.
E.g. Array, Stack, Queue, linked list	E.g. Tree, Graph, Map

Lists

- A list is a **finite**, **ordered** sequence of data items.
- Two standard approach to implement list - **Arrays**, **Linked list**
- **Arrays** – static data structure
- **Linked list** – dynamic data structure

ARRAYS

An array is a **finite**, **ordered** and collection of **homogeneous** data elements.

int A[100]

Size : Number of elements in an array.

Type : data type

Base : base address

Index : subscript used to refer array elements

Range of Index : Indices change from lower bound **LB** to and upper bound **UB**

Arrays

Actual Address of the **1st** element of the array is known as **Base Address (B)**
Here it is 1100

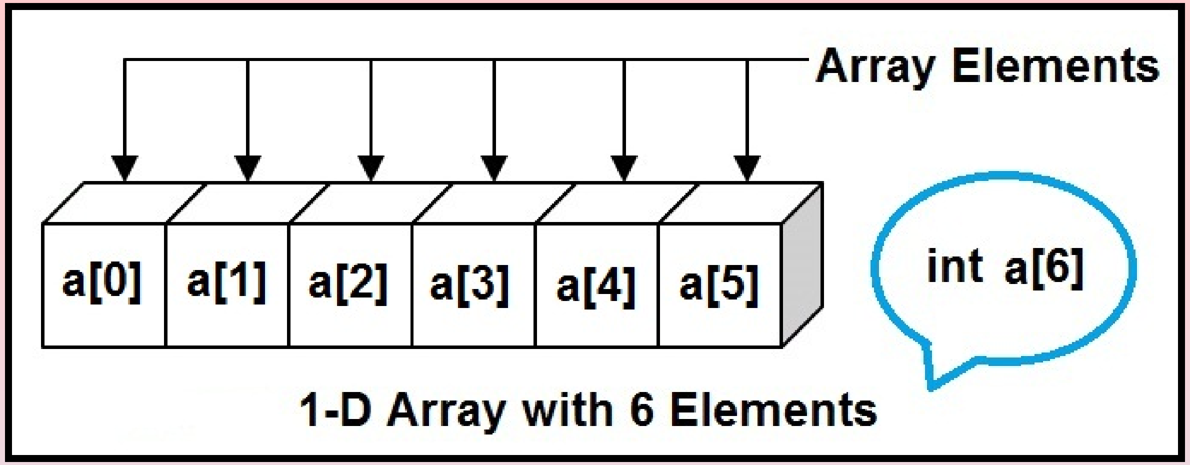
Memory space acquired by every element in the Array is called **Width (W)**
Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

Lower Limit/Bound of Subscript **(LB)**

Arrays

1 D - Arrays



Indexing formula

$$\textbf{Address}\ (A[i]) = \mathbf{B} + (\mathbf{i} - \mathbf{LB}) \times w$$

B = Base address
w = Storage Size of one element stored in the array (in byte)
i = Subscript of element whose address is to be found
LB = Lower limit / Lower Bound of subscript, if not specified assume 0 (zero)

Arrays

Given the base address of an array B[1300…..1900] as 1020 and size of each element is 2 bytes in the memory. Find the address of B[1700].

Solution:

The given values are: B = 1020, L B= 1300, W = 2, i = 1700

Address of A [I] = B + W * (i – LB)

= 1020 + 2 * (1700 – 1300)

= 1020 + 2 * 400

= 1020 + 800

= 1820

Operations on Arrays

- ❖Traversing
- ❖Sorting
- ❖Searching
- ❖Insertion
- ❖Deletion

8	6	5	4	2	1	9	7	3	6	4	2

Arrays

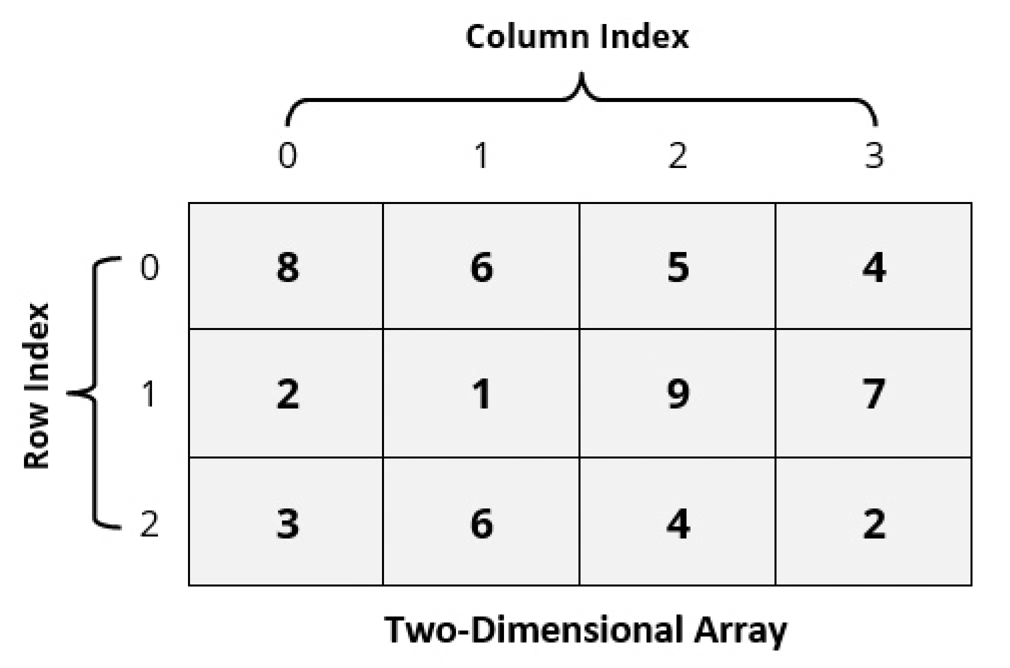
Two dimensional Arrays

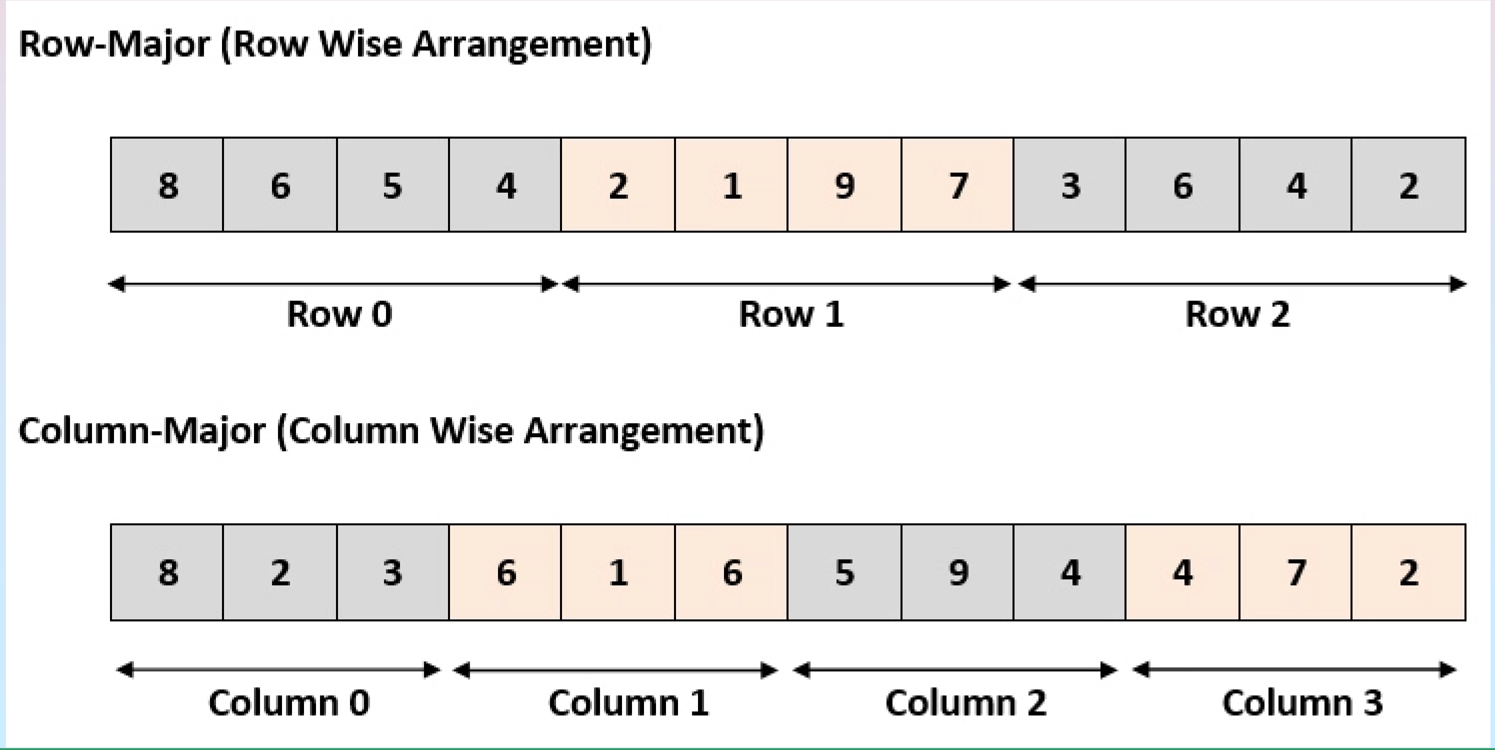
a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Memory Representation

1. **Row-major form**
2. **Column major form**

Arrays


Column Index
0 1 2 3
Row Index
0 8 6 5 4
1 2 1 9 7
2 3 6 4 2
Two-Dimensional Array


Row-Major (Row Wise Arrangement)
8 6 5 4 2 1 9 7 3 6 4 2
Row 0
Row 1
Row 2
Column-Major (Column Wise Arrangement)
8 2 3 6 1 6 5 9 4 4 7 2
Column 0
Column 1
Column 2
Column 3

Arrays

The address of a location in Row Major System is calculated using the following formula:

$$\text{Address of } A[I][J] = B + w * [N * (I - Lr) + (J - Lc)]$$

Address of A [I][J] Column Major Wise = $B + w * [(I - Lr) + M * (J - Lc)]$

B = Base address

I = Row subscript of element whose address is to be found

J = Column subscript of element whose address is to be found

W = Storage Size of one element stored in the array (in byte)

Lr = Lower limit of row/start row index of matrix, if not given assume 0 (zero)

Lc = Lower limit of column/start column index of matrix, if not given assume 0 (zero)

M = Number of row of the given matrix

N = Number of column of the given matrix

END