Unit-1

Introduction of Data Structure

Introduction of Data

- **Data** is distinct pieces of information.
- Data can exist in a variety of forms:
  - Numbers or texts on pieces of paper
  - Bits or bytes stored in electronic memory
  - Facts stored in human's mind
- People have used the word **DATA** to mean computer information that is transmitted or stored.
- Strictly speaking data is a **plural of datum**, a single piece of information**.**
- In practice, however, people use data as both the singular and plural form of the word.

Introduction of Data Structure

- **Data structure** is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way.

- Data structure is about rendering (providing) data elements in terms of some relationship, for better organization and storage.

Introduction of Data Structure

- **For example,** we have data player's name "Hitesh" and age 26.
- Here we have two data elements:
  - Player's name: string datatype
  - Player's age: integer datatype
- We can organize this data as a record like Player record.
- Now we can collect and store Player's records in a file or database as a data structure.
- For example, "Gayle" 30, "Sachin" 31, "Parth" 33.
- In simple language, Data structures are structures programmed to store ordered data, so that various operations can be performed easily.

Introduction of Data Structure

- Anything that can store data can be called as a data structure, hence Integer, Float, Boolean, Char etc… are also called data structures. They are also known as **Primitive Data Structures/ In-Built Data Structures**.

- Complex type of data structures are also available, which are used to store large and connected data. They are also known as **Non-Primitive Data Structures/ Derived Data Structures/ Abstract Data Structures**. For example, Array, Stack, Queue, Linked list, Tree, Graph etc….

Introduction of Data Structure

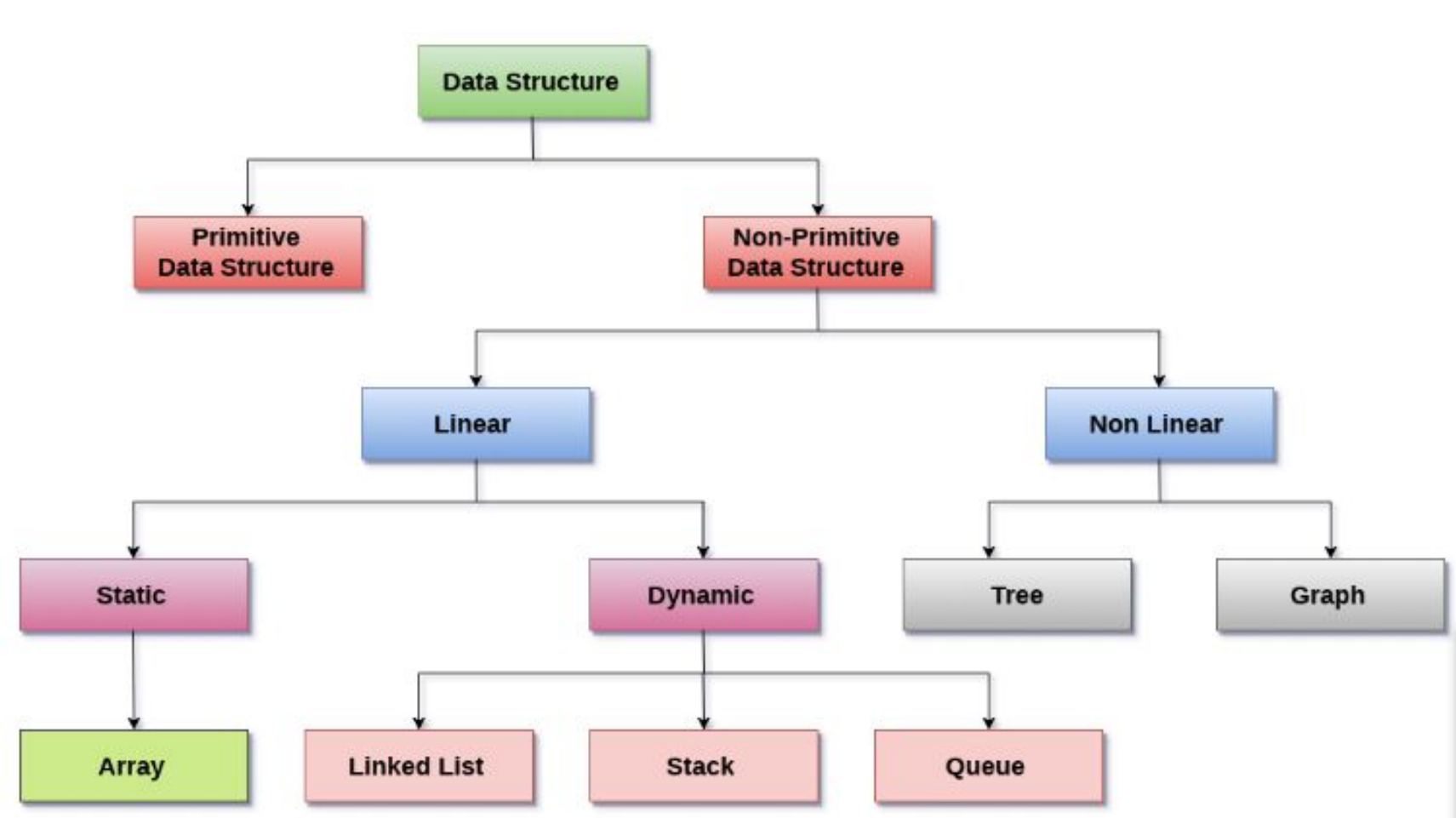


Figure : Classification of Data Structures

Introduction of Data Structure

- The data structures can also be classified on the basis of the following characteristics:

- **Linear Data Structures:** A data structure is called linear if all of its elements are arranged in the linear order. In linear data structures, the elements are stored in non-hierarchical way where each element has the successors and predecessors except the first and last element

Introduction of Data Structure

Types of Linear Data Structures are given below:

- **Arrays:** An array is a collection of similar type of data items and each data item is called an element of the array. The data type of the element may be any valid data type like char, int, float or double.

- **Linked List:** Linked list is a linear data structure which is used to maintain a list in the memory. It can be seen as the collection of nodes stored at non-contiguous memory locations. Each node of the list contains a pointer to its adjacent node.

Introduction of Data Structure

- **Stack:** Stack is a linear list in which insertion and deletions are allowed only at one end, called **top**. A stack is an abstract data type (ADT), can be implemented in most of the programming languages. It is named as stack because it behaves like a real-world stack, for example: - piles of plates or deck of cards etc.

- **Queue:** Queue is a linear list in which elements can be inserted only at one end called **rear** and deleted only at the other end called **front**. It is an abstract data structure, similar to stack. Queue is opened at both end therefore it follows First-In-First-Out (FIFO) methodology for storing the data items.

Introduction of Data Structure

- **Non Linear Data Structures:** This data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in sequential structure.

Types of Non Linear Data Structures are given below:

- **Trees:** Trees are multilevel data structures with a hierarchical relationship among its elements known as nodes. The bottommost nodes in the hierarchy are called **leaf node** while the topmost node is called **root node**. Each node contains pointers to point adjacent nodes.

Introduction of Data Structure

- **Graphs:** Graphs can be defined as the pictorial representation of the set of elements (represented by vertices) connected by the links known as edges. A graph is different from tree in the sense that a graph can have cycle while the tree can not have the one.

Operations on Data Structures

1) **Traversing:** Every data structure contains the set of data elements. Traversing the data structure means visiting each element of the data structure in order to perform some specific operation like searching or sorting.

2) **Insertion:** Insertion can be defined as the process of adding the elements to the data structure at any location.

3) **Deletion:** The process of removing an element from the data structure is called Deletion. We can delete an element from the data structure at any random location.

Operations on Data Structures

4) **Searching:** The process of finding the location of an element within the data structure is called Searching. There are two algorithms to perform searching, Linear Search and Binary Search.

5) **Sorting:** The process of arranging the data structure in a specific order is known as Sorting. There are many algorithms that can be used to perform sorting, for example, insertion sort, selection sort, bubble sort, etc.

6) **Merging:** When two lists List A and List B of size M and N respectively, of similar type of elements, clubbed or joined to produce the third list, List C of size (M+N), then this process is called merging

Data Representation

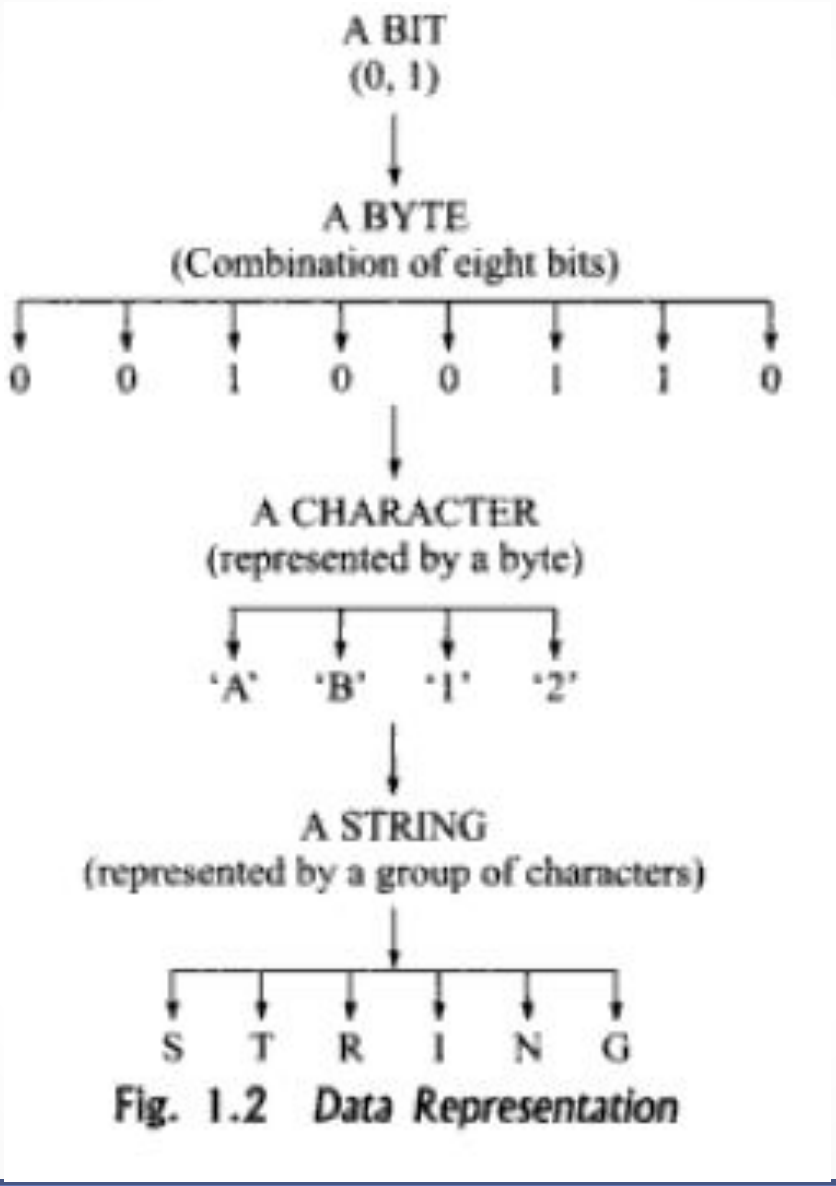


Fig. 1.2 *Data Representation*

Data Representation

- Various methods are used to represent data in computers.
- Hierarchical layers of data structure are used to make the use of data structure easy and efficient.
- The basic unit of a data representation is a bit. The value of bit asserts one of the two mutually exclusive possibilities – 0 or 1.
- 8 bits together form one byte which represents a character and one or more than one characters are used to form a string.

Data Representation

- Integer Representation :
  - Integer is a basic data type which is used to store negative and non-negative values.
  - Non-negative value can be represented by binary representation.
  - For negative binary numbers the methods of representation used are one's complement and two's complement.

- Real Number Representation :
  - The method used to represent real numbers in computers is floating – point notation.
  - In this notation, the real number is represented by a number called a mantissa, times a base raised to an integer power, called an exponent.

Data Representation

- Character Representation :
  - The information in computers is not always interpreted numerically.
  - For e.g. to store name of the employee which requires a different data representation that is in character string form.
  - There are different codes available to store data in character form such as BCD, EBCDIC and ASCII.
  - E.g. if 8 bits are used to represent a character, then up to $2^8 = 256$ different characters can be represented as bit patterns.

Abstract Data Type

ADTs are like user defined data types which defines operations on values using functions without specifying what is there inside the function and how the operations are performed.

Abstract data type is specification of the data type which specifies the logical and mathematical model of the data type.

An abstract data type, or ADT, specifies a set of operations (or methods) and the semantics of the operations (what they do), but it does not specify the implementation of the operations. That's what makes it abstract.

Example:

Stack ADT

Abstract Data Type

Operations:

push() - Insert Element into stack

pop() - Delete Element from stack

isEmpty() - check if stack is empty

isFull() - check if stack is full

- Think of ADT as a black box which hides the inner structure and design of the data type from the user.
- There are multiple ways to implement an ADT.
- A Stack ADT can be implemented using arrays or linked list.
- ADT Provides Abstraction.

Introduction of Datatype

- **Data types** specify how we enter data into our programs and what type of data we enter.
- C language has some predefined set of data types to handle various kinds of data that we can use in our program.
- C data types are used to:
  - Identify the **type of the variable** when it is declared.
  - Identify the **type of the return value of a function**.
  - Identify the **type of the parameter expected by a function**.

Primary Datatype

Table : Primary data types in C

Data Type	Keyword used	Size in bytes	Range	Use
Character	char	1	-128 to 127	To store characters
Integer	int	2	-32768 to 32767	To store integer numbers
Floating point	float	4	3.4E -38 to 3.4E +38	To store floating point numbers
Double	double	8	1.7E -308 to 1.7E +308	To store big floating point numbers
Valueless	void	0	Valueless	-----

Data Structure and Structured Type

A structured type refers to a data structure which is made-up of one or more elements known as components. These elements are simpler data structure that exist in language.

The components of structured data type are grouped together according to a set of rules. for example the representation of polynomials requires at least two components.

- Coefficient
- Exponent

The structure type is implemented using **struct** keyword.

Atomic Type

Generally, a data structure is represented by a memory block, which has two parts

- Data Storage
- Address Storage

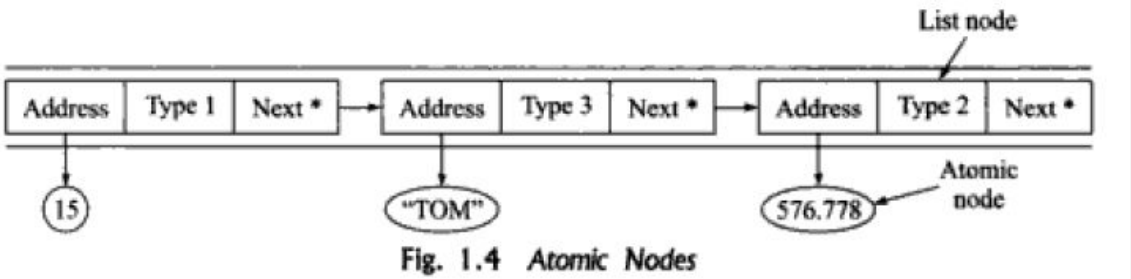


Fig. 1.4 *Atomic Nodes*

This facilitates in storing data and relating it to some other data by means of storing pointer in the address part. data type that is indivisible or cannot be further subdivided into smaller components

An atomic type data is data structure that contains only the data items and not the pointer. Thus, for a list of data items, several atomic type nodes may exist each with single data item corresponding to one of the legal data types. the list maintained using a list node which contains pointer to these atomic nodes and a type indicator indicating the type of atomic node to which it points.

Difference between Abstract Data Type, Data Types and Data Structures

Abstract data type is specification of the data type which specifies the logical and mathematical model of the data type.

Data type is implementation of abstract data type.

Data Structure refers to collection of computer variables that are connected in some specific manner.

Algorithm

- **“Algorithm" : It is a step by step descriptive procedure for solving a particular problem.**

- The algorithm gives logic of the program, that is, a step-by-step description of how to arrive at a solution.

A sequence of instructions must process the following characteristics:

- Instructions must be clear
- Instructions must be effective.
- Not even a single instruction must not be repeated infinitely
- After the algorithm gets terminated, the desired result must be obtained

Different Approaches to Designing an Algorithm

1. Top-Down Approach
2. Bottom-Up Approach

Top-Down Approach

Top-Down Model is a system design approach where design starts from the system as a whole. Complete System is then divided into smaller sub-applications with more details. Each part again goes through the top-down approach till the complete system is designed with all minute details. Top Down approach is also termed as breaking the bigger problem into smaller problems and solving them individually in recursive manner.

Different Approaches to Designing an Algorithm

Bottom-Up Approach

- Bottom-Up Model is a system design approach where parts of the system are defined in details. Once these parts are designed and developed, then these parts or components are linked together to prepare a bigger component. This approach is repeated until the complete system is built. Advantage of Bottom-Up Model is in making decisions at very low level and to decide the re-usability of components.

Divide – and – Conquer approach

- The **Divide and Conquer** strategy involves dividing the problem into sub-problem, recursively solving them, and then recombining them for the final answer.
- E.g. Merge

Different Approaches to Designing an Algorithm

Sr. No.	Key	Bottom-Up Model	Top-Down Model
1	Focus	In Bottom-Up Model, the focus is on identifying and resolving smallest problems and then integrating them together to solve the bigger problem.	In Top-down Model, the focus is on breaking the bigger problem into smaller one and then repeat the process with each problem.
2	Language	Bottom-Up Model is mainly used by object oriented programming languages like Java, C++ etc.	Top-Down Model is followed by structural programming languages like C, Fortran etc.
3	Redundancy	Bottom-Up model is better suited as it ensures minimum data redundancy and focus is on re-usability.	Top-down model has high ratio of redundancy as the size of project increases.
4	Interaction	Bottom-Up model have high interactivity between various modules.	Top-down model has tight coupling issues and low interactivity between various modules.
5	Approach	Bottom-up model is based on composition approach.	Top-down model is based on decomposition approach.
6	Issues	In Bottom-Up, some time it is difficult to identify overall functionality of system in initial stages.	In Top-Down, it may not be possible to break the problem into set of smaller problems.

Complexity of Data Structure

Complexity is a rough approximation of the number of steps necessary to execute an algorithm.

Two types of Complexity of Data Structure

1. Time Complexity
2. Space Complexity

Complexity of Data Structure

1. Time Complexity

The time complexity of an algorithm is basically the running time of a program as a function of the input size.

```
for(i=0; i < N; i++)
{
   statement;
}
```

The time complexity for the above algorithm will be Linear. The running time of the loop is directly proportional to N. When N doubles, so does the running time.

Complexity of Data Structure

2. Space Complexity

- Space complexity of an algorithm represents the amount of memory space needed the algorithm in its life cycle.
- Space needed by an algorithm is equal to the sum of the following two components
- A fixed part that is a space required to store certain data and variables (i.e. simple variables and constants, program size etc.), that are not dependent of the size of the problem.
- A variable part is a space required by variables, whose size is totally dependent on the size of the problem. For example, recursion stack space, dynamic memory allocation etc.

Analysis of Algorithm

We can analyze algorithm in Three Different ways as follows.

1. Best Case
2. Worst Case
3. Average Case

Analysis of Algorithm

1. **Best Case**

The best-case complexity of the algorithm is the function defined by the minimum number of steps taken on any instance of size n.

The term ‘best-case performance’ is used to analyse an algorithm under optimal conditions.

Analysis of Algorithm

2. Worst Case

The worst-case complexity of the algorithm is the function defined by the maximum number of steps taken on any instance of size n.

The worst-case running time of an algorithm is an upper bound on the running time for any input.

Therefore, having the knowledge of worst-case running time gives us an assurance that the algorithm will never go beyond this time limit.

Analysis of Algorithm

3. Average Case

The average-case running time of an algorithm is an estimate of the running time for an ‘average’ input.

It specifies the expected behaviour of the algorithm when the input is randomly drawn from a given distribution.

Asymptotic Notations

- Different types of asymptotic notations are used to represent the complexity of an algorithm.
- Following asymptotic notations are used to calculate the running time complexity of an algorithm.
  - **O** − Big Oh
  - **Ω** − Big omega
  - **θ** − Big theta
  - **o** − Small Oh
  - **ω** − Small omega

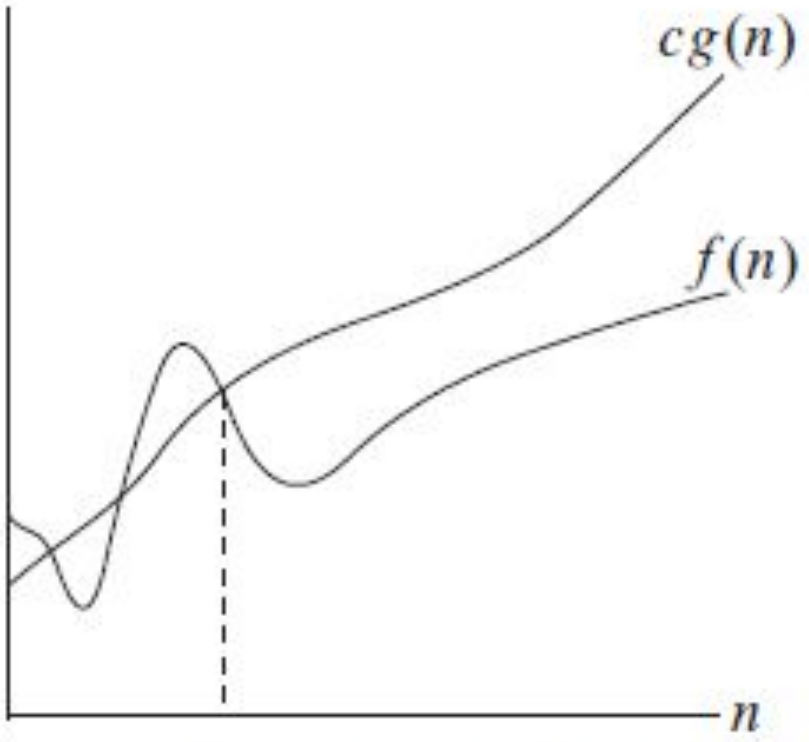
O – Big Oh

- Big – Oh Notation :
  - In this f(x) = O(g(x)). It means the growth rate if f(x) is asymptotically less than or equal to the growth rate of g(x).
  - g(n) is an asymptotic upper bound for f(n).

O – Big Oh

Example

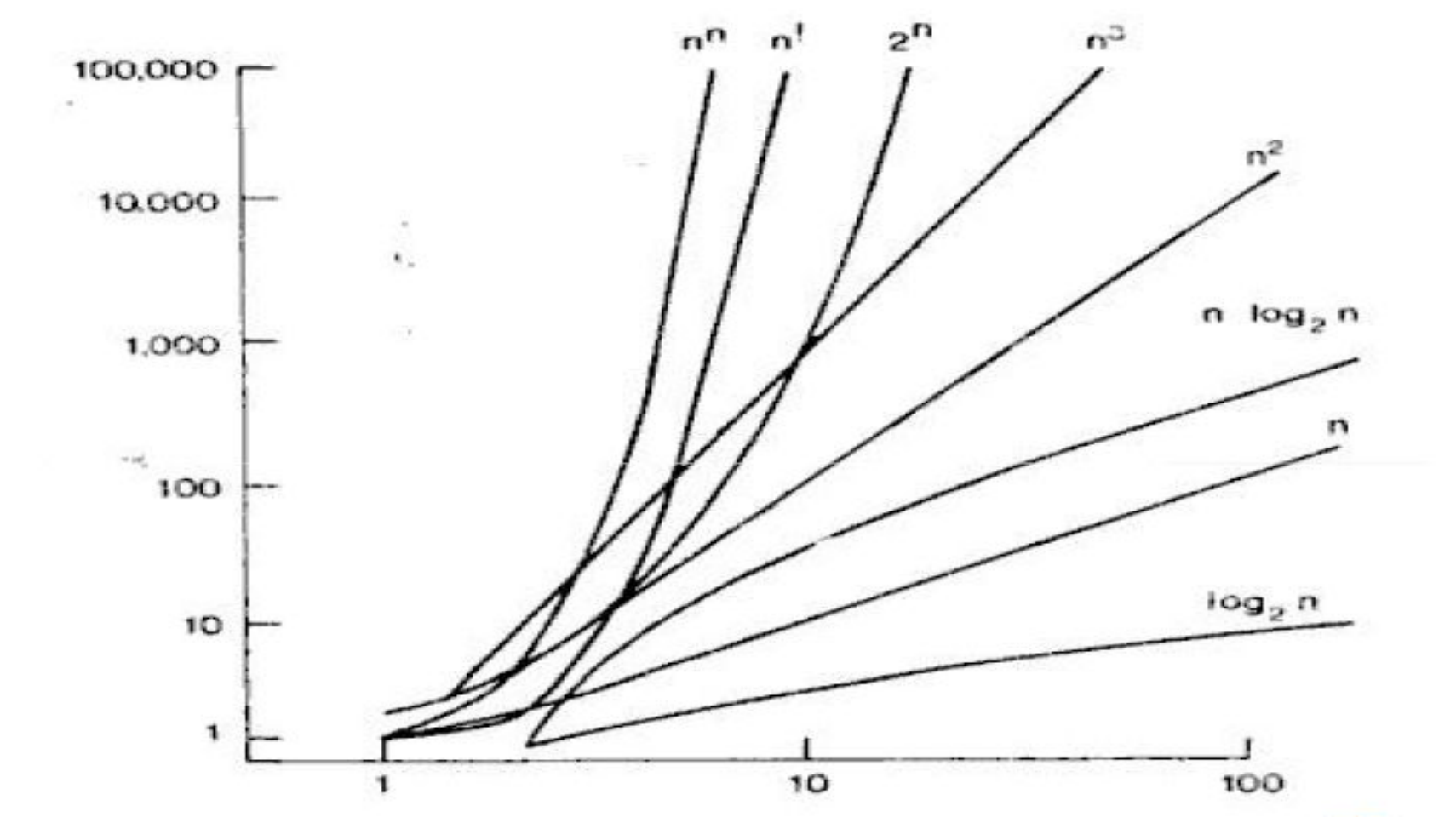
- f(n)=2n+3
- cg(n)=10n



$f(n) = O(g(n))$

(b)

O – Big Oh



O – Big Oh

- O(1) – Constant time :
  - This means that the algorithm requires the same fixed no. of steps regardless of the size of the task.
  - E.g :-
    - Insert and Remove operations for a queue
    - Any Mathematical Expression i.e. sum=a+b

O – Big Oh

- O(n) – Linear time :
  - This means that the algorithm requires a no. of steps proportional to the size of task.
  - E.g. :-
    - Traversal of a list (a linked list or an array) with n elements.

O – Big Oh

- $O(n^2)$ – Quadratic time :
  - The no. of operations is proportional to the size of the task squared.
  - E.g. :-
    - Comparing two 2-dimensional arrays of size n by n.

O – Big Oh

- O(log n) – Logarithmic time :
  - E.g. :-
    - Binary search in a sorted list of n elements.

O – Big Oh

- O(n log n) – “n log n” time :
  - E.g. :-
    - More advanced sorting algorithms – quick sort, merge sort.
- $O(a^n)$ (a>1) : Exponential time :
  - E.g. :-
    - Recursive Fibonacci implementation

O – Big Oh

- Polynomial growth (linear , quadratic, cubic, etc.) is considered manageable as compared to exponential growth.
- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(a^n)$

O – Big Oh

- **Calculating complexity : E.g. :-**

```
void main()
{
   int a = 10,I,j; // O(1)
   for (i=0;i<n;i++) // O(n)
   {
     for(j=0;j<I;j++) // O(n)
     {
          printf("a = %d",a); //O(1)
     }
   }
}
```

Total execution time = $1 + n + n^2 + 1 = O(n^2)$

Thank You...!!!!