Data Structures

Data Structure

- Data Structure can be defined as the group of data elements which provides an efficient way of storing and organizing data in the computer so that it can be used efficiently.
- A data structure is a specialized format for organizing and storing data. Structures are arrays, Linked List, Stack, Queue, etc.
- Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways

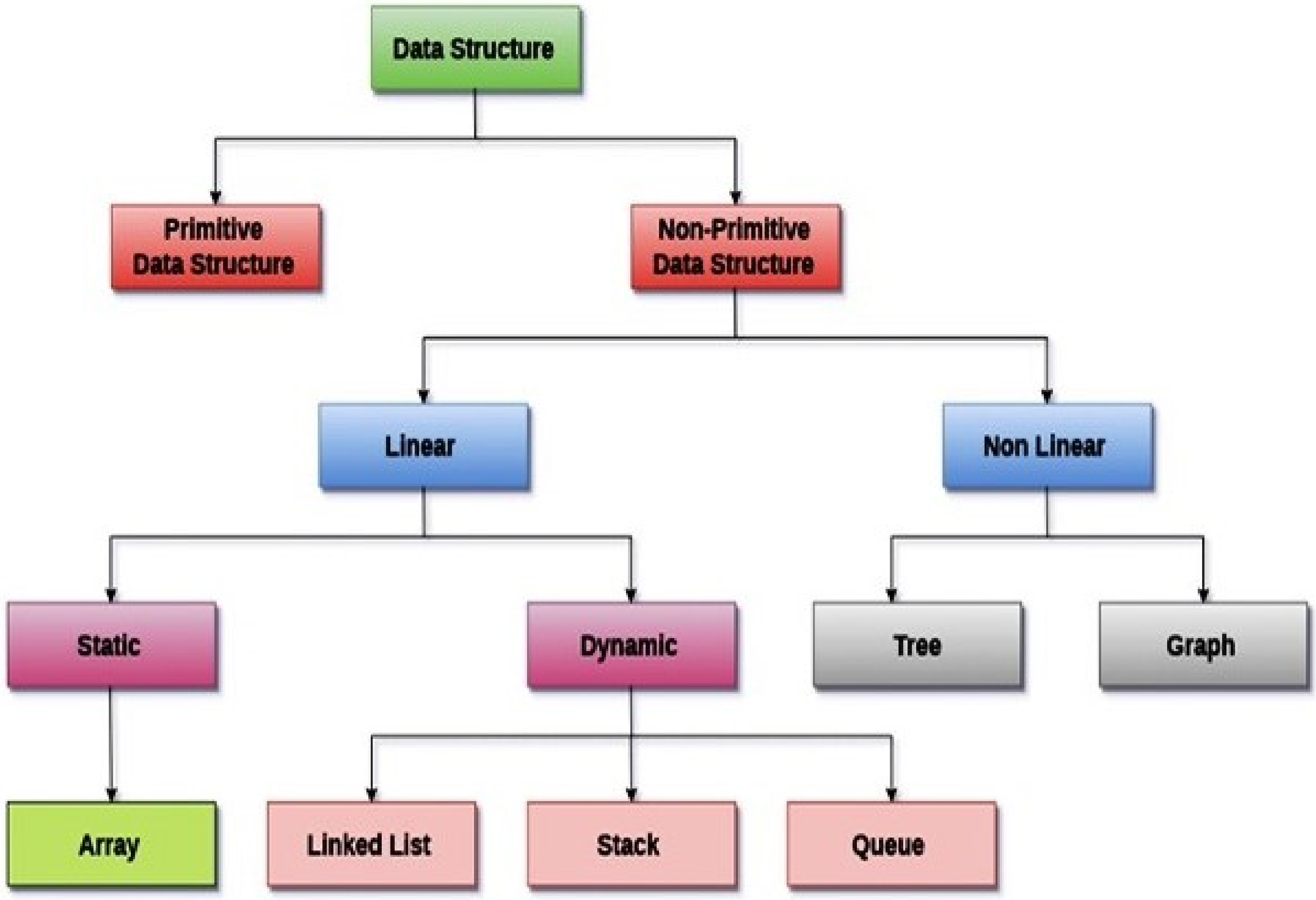
Need for data structure:

- A data structure helps you to understand the relationship of one data element with the other and organize it within the memory. Sometimes the organization might be simple. For example

1. List of names of months in a year –Linear Data Structure.
2. List of historical places in the world- Non-Linear Data Structure.

   A data structure helps you to analyze the data, store it and organize it in a logical and mathematical manner.

Types of Data Structure



Primitive Data Structures

- Primitive Data Structures are the basic data structures that directly operate upon the machine instructions.
- They have different representations on different computers.
- For example, integer, character, and string are all primitive data types.
- Programmers can use these data types when creating variables in their programs.

Non-primitive Data Structures

- Non-primitive data structures are more complicated data structures and are derived from primitive data structures.
- They are of Two types:

1. Linear
2. Non-Linear

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 nonhierarchical way where each element has the successors and predecessors except the first and last element.

Linear Data Structures Cont..

Types of Linear Data Structures are given below:

1. **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.
- The elements of array share the same variable name but each one carries a different index number known as subscript.
- The array can be one dimensional, two dimensional or multidimensional.

Linear Data Structures Cont..

2. 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.

Linear Data Structures Cont..

3. 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.

Linear Data Structures Cont..

4. 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.

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
- Examples are Tree and Graphs

Non-Linear Data Structures Cont..

1. 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.
- Tree data structure is based on the parent-child relationship among the nodes.
- Each node contains pointers that points to the child node.
- Each node in the tree can have more than one children except the leaf nodes whereas each node can have at most one parent except the root node.
- Trees can be classified into many categories which will be discussed later.

Non-Linear Data Structures Cont..

2. 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 cannot have the one.

Algorithm

- An algorithm is a procedure having well defined steps for solving a particular problem.
- Algorithm is finite set of logic or instructions, written in order for accomplish the certain predefined task.
- It is not the complete program or code, it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudo code.
- Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language

Characteristics of an Algorithm

An algorithm must follow the mentioned below characteristics:

- **Input:** An algorithm must have 0 or well defined inputs.
- **Output:** An algorithm must have 1 or well defined outputs, and should match with the desired output.
- **Feasibility:** An algorithm must be terminated after the finite number of steps.
- **Independent:** An algorithm must have step-by-step directions which is independent of any programming code.
- **Unambiguous:** An algorithm must be unambiguous and clear. Each of their steps and input/outputs must be clear and lead to only one meaning.

Algorithm Cont..

From the data structure point of view, following are some important categories of algorithms –

- **Search** – Algorithm to search an item in a data structure.
- **Sort** – Algorithm to sort items in a certain order.
- **Insert** – Algorithm to insert item in a data structure.
- **Update** – Algorithm to update an existing item in a data structure.
- **Delete** – Algorithm to delete an existing item from a data structure.

Time and space complexity of Algorithms

- Sometimes, there are more than one way to solve a problem. We need to learn how to compare the performance of different algorithms and choose the best one to solve a particular problem.
- While analysing an algorithm, we mostly consider time complexity and space complexity.
- **Time complexity** of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.
- **Space complexity** of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input. Space complexity is required in situations when limited memory is available

Asymptotic Analysis

- Asymptotic analysis of an algorithm refers to defining the mathematical foundation/framing of its run-time performance.
- Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.
- Usually, the time required by an algorithm falls under three types

1. **Best Case** − Minimum time required for program execution.
2. **Average Case** − Average time required for program execution.
3. **Worst Case** − Maximum time required for program execution.

Asymptotic Notations

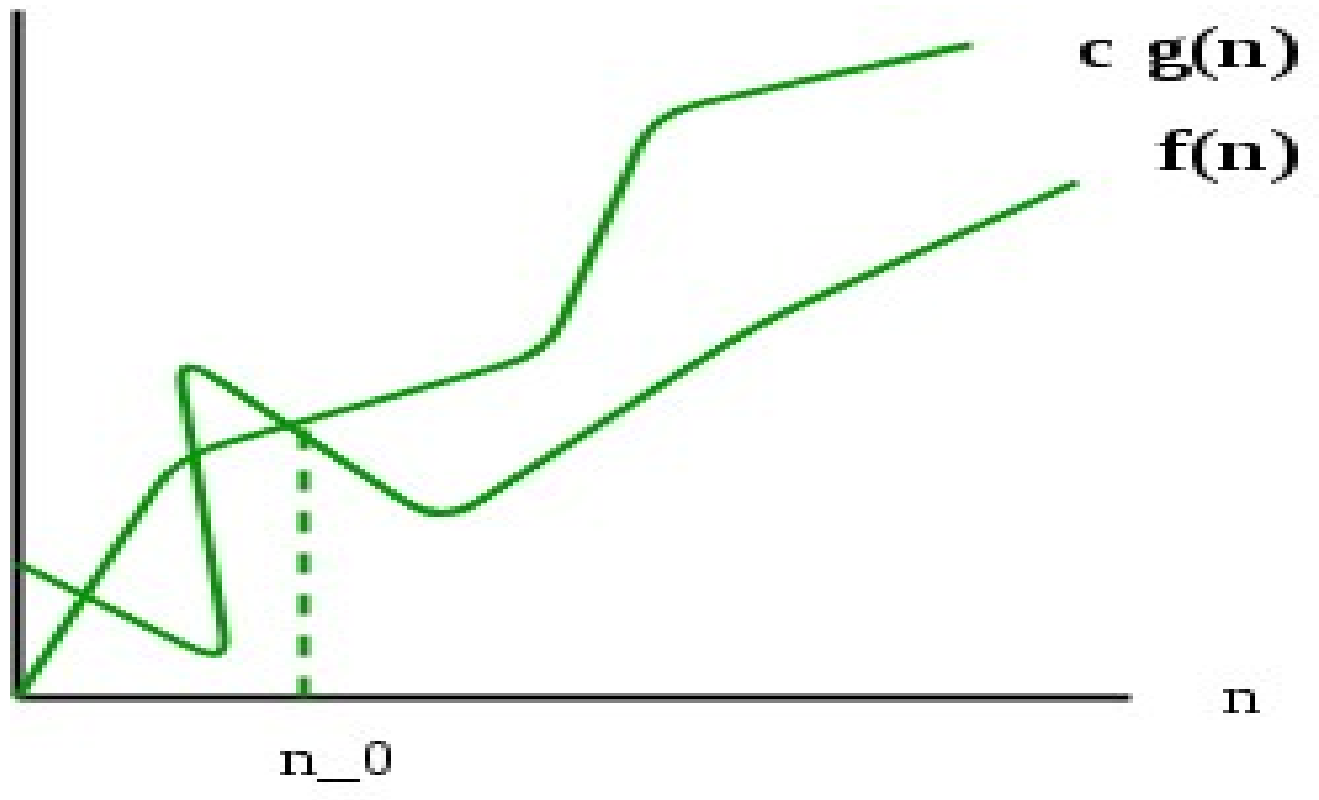
- Asymptotic Notations are mathematical tools used to analyze the performance of algorithms by understanding how their efficiency changes as the input size grows.
- Asymptotic analysis allows for the comparison of algorithms' space and time complexities by examining their performance characteristics as the input size varies.
- Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.
- O Notation
- Ω Notation
- θ Notation

Asymptotic Notations Cont..

Big O Notation:

- It was introduced by Paul Bachmann in 1894.
- The Big O notation defines an upper bound of an algorithm.
- *Big-O(Worst Case) It is defined as the condition that allows an algorithm to complete statement execution in the longest amount of time possible.*
- *The maximum time required by an algorithm or the worst-case time complexity.*
- *it returns the highest possible output value(big-O) for a given input.*

Big O Notation:



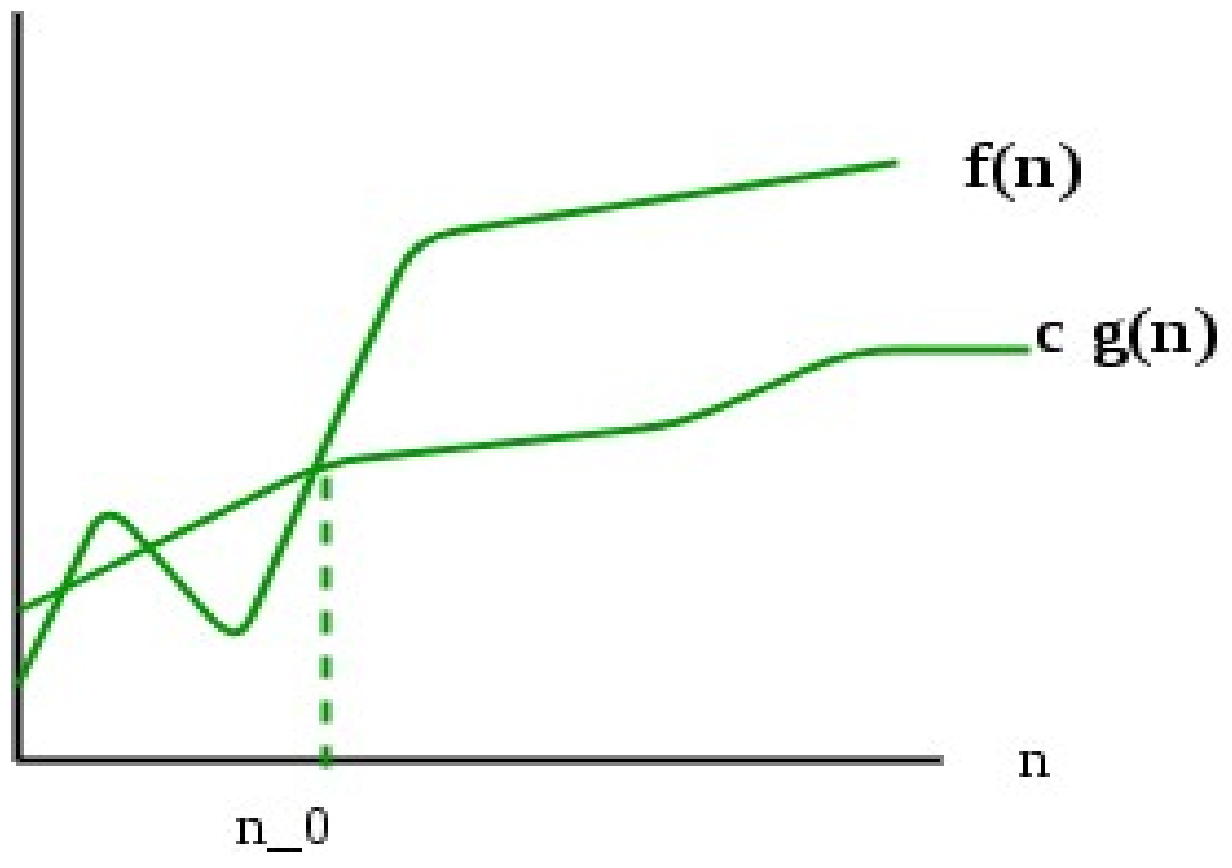
$$f(n) = O(g(n))$$

Asymptotic Notations Cont..

Omega_Notation(Ω-Notation):

- just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.
- *thus, it provides the best case complexity of an algorithm.*
- *It is defined as the condition that allows an algorithm to complete statement execution in the shortest amount of time.*

Omega Notation:



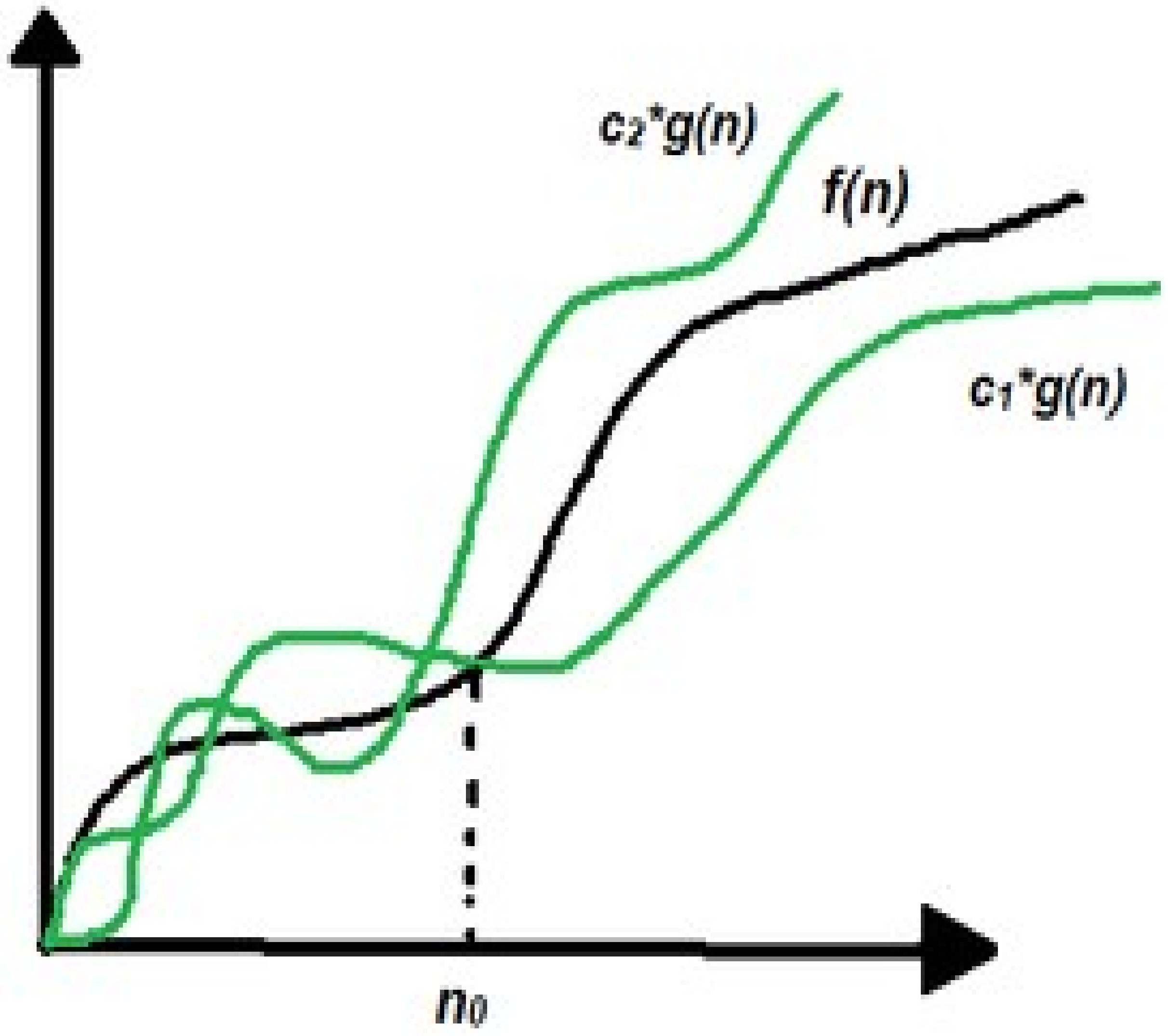
f(n) = Omega(g(n))

Asymptotic Notations Cont..

Theta Notation(Θ Notation):

- The theta notation bounds a function from above and below, so it defines exact asymptotic behavior.
- *Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the* ***average-case*** *complexity of an algorithm.*
- *Theta (Average Case) You add the running times for each possible input combination and take the average in the average case.*

Theta Notation:



$c_2 * g(n)$
$f(n)$
$c_1 * g(n)$
n_0