Q.1. What is data structure and what are different types of it?

OR

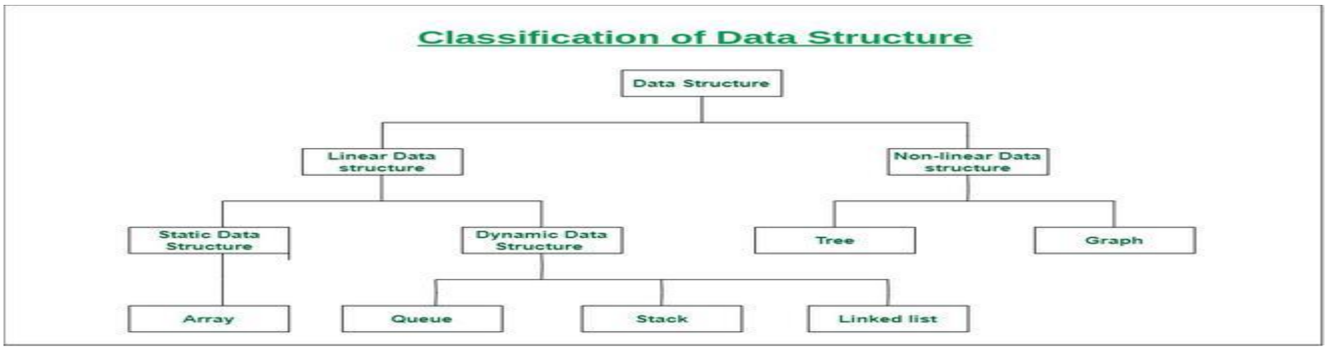
Q.2. Discuss primitive and non primitive Data Structure.

Ans:

Data Structure is the way of organizing and storing data in a computer system so that it can be used efficiently. Data structures help in storing, organizing, and analyzing the data in a logical manner. The following points highlight the need of data structures in computer science:

1. It depicts the logical representation of data in computer memory.
2. It represents the logical relationship between the various data elements.
3. It helps in efficient manipulation of stored data elements.
4. It allows the program to process the data in an efficient manner.

Types of Data Structure



Data structures are primarily divided into two classes, primitive and non-primitive.

I. **Primitive data structures** include all the fundamental data structures that can be directly manipulated by machine-level instructions.

Some of the common primitive data structures include integer, character, real, boolean, etc.

II. **Non-primitive data structures**, on the other hand, refer to all those data structures that are derived from one or more primitive data structures. The objective of creating non-primitive data structures is to form sets of homogeneous or heterogeneous data elements.

Non-primitive data structures are further categorized into two types:

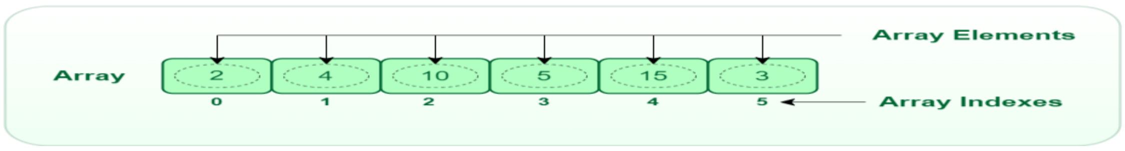
A. Linear Data Structure

B. Non-linear Data Structure

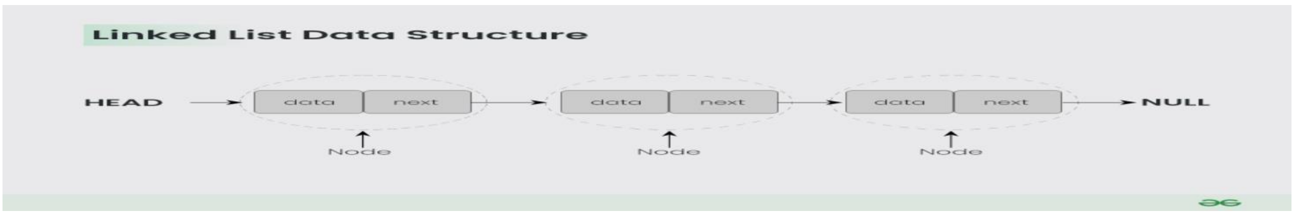
In linear data structures, all the data elements are arranged in a linear or sequential fashion.

Examples of linear data structures include:

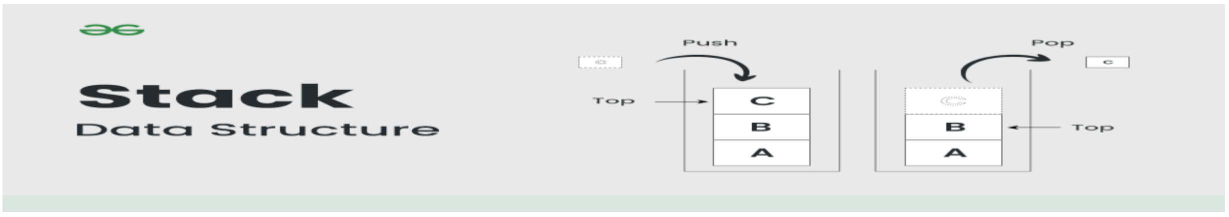
1. **Array**: It is collection of elements of similar data types stored at contiguous location.



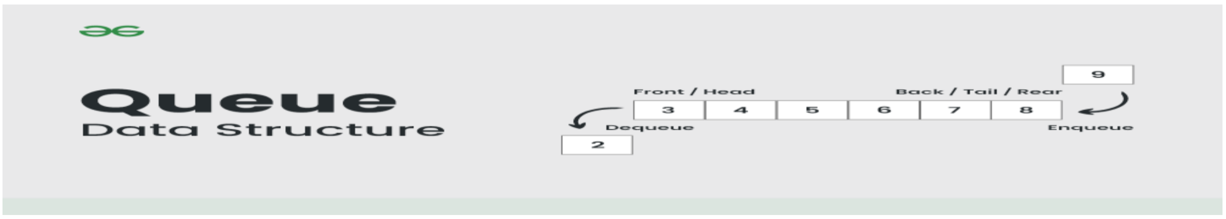
2. **Linked Lists**: It is a collection of "nodes" connected together via links. These nodes consist of the data to be stored and a pointer to the address of the next node within the linked list.



3. **Stacks:** Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last.



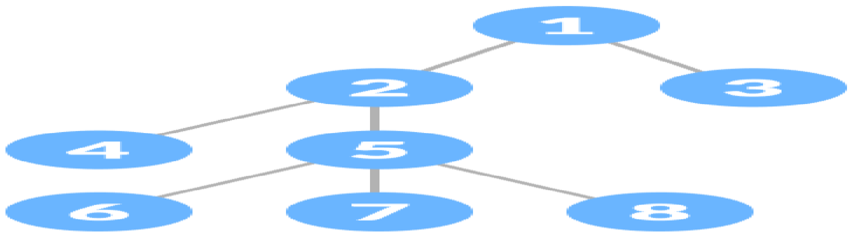
4. **Queues:** A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order.



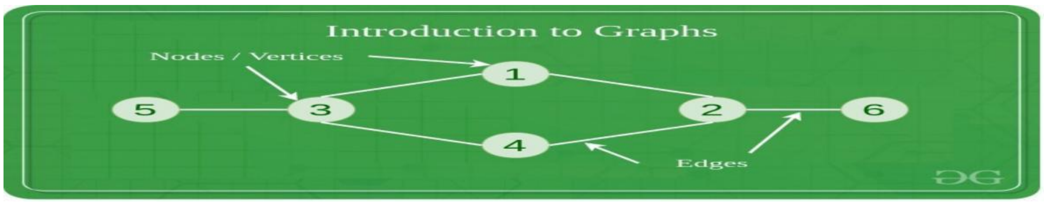
In non-linear data structures, there is no definite order or sequence in which data elements are arranged. For instance, a non-linear data structure could arrange data elements in a hierarchical fashion.

Examples of non-linear data structures are:

1. Tree:A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes. The topmost node of the tree is called the root, and the nodes below it are called the child nodes. Each node can have multiple child nodes, and these child nodes can also have their own child nodes, forming a recursive structure.



2. Graph:A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(V) and a set of edges(E). The graph is denoted by G(E, V).



Q.3. Explain complexity of algorithm.

Ans: An algorithm can be defined as a step by step procedure that provides solution to a given problem. It comprises of a well-defined set of finite number of steps or rules that are executed sequentially to obtain the desired solution.

An algorithm solves only a single problem at a time. However, the same problem can be solved using multiple algorithms. The benefit of using multiple algorithms to solve the same problem is purely situational. can represent an algorithm in a number of ways, right from normal English language phrases to graphical representation using flow charts. However, such representations are mainly useful when the
algorithm is simple and small.

Another way of representing an algorithm is the pseudocode. Pseudocode is an informal representation of the algorithm that provides a complete outline of a program so that the programmers can easily understand it and transform it into a program using the programming language of their choice.

An algorithm's complexity is a measure of the amount of data that it must process in order to be efficient.

1. Time Complexity

Time complexity is defined in terms of how many times it takes to run a given algorithm, based on the length of the input. Time complexity is not a measurement of how much time it takes to execute a particular algorithm because such factors as programming language, operating system, and processing power are also considered.

Time complexity is a type of computational complexity that describes the time required to execute an algorithm. The time complexity of an algorithm is the amount of time it takes for each statement to complete. As a result, it is highly dependent on the size of the processed data. It also aids in defining an algorithm's effectiveness and evaluating its performance.

2. **Space Complexity**

When an algorithm is run on a computer, it necessitates a certain amount of memory space. The amount of memory used by a program to execute it is represented by its space complexity. Because a program requires memory to store input data and temporal values while running, the space complexity is auxiliary and input space.

Time Complexity	**Space Complexity**
Calculates the time required	Estimates the space memory required
Time is counted for all statements	Memory space is counted for all variables, inputs, and outputs.
The size of the input data is the primary determinant.	Primarily determined by the auxiliary variable size
More crucial in terms of solution optimization	More essential in terms of solution optimization

Q.4. Illustrate the term algorithm with its representation.

OR

Q.5. List the characteristics of Algorithm

Ans:

An algorithm can be defined as a step by step procedure that provides solution to a given problem. It comprises of a well-defined set of finite number of steps or rules that are executed sequentially to obtain the desired solution.

An algorithm solves only a single problem at a time. However, the same problem can be solved using multiple algorithms. The benefit of using multiple algorithms to solve the same problem is purely situational. can represent an algorithm in a number of ways, right from normal English language phrases to graphical representation using flow charts. However, such representations are mainly useful when the
algorithm is simple and small.

Another way of representing an algorithm is the pseudocode. Pseudocode is an informal representation of the algorithm that provides a complete outline of a program so that the programmers can easily understand it and transform it into a program using the programming language of their choice.

While representing an algorithm in pseudocode form, you must use certain conventions consistently throughout the algorithm. This helps in easy understanding of the algorithm. Following are some of the general conventions that are followed while writing pseudocode:

1. Provide a valid name for the algorithm written using pseudocode.
2. For each line of instruction, specify a line number.
3. Always begin a identifier name with English alphabet.
4. It is not necessary to explicitly specify the data type of the variables.
5. Always indent the statements present inside a block structure appropriately.
6. Use read and write instructions to specify input and output operations respectively.
7. Use if or if else constructs for conditional statements.

An algorithm's complexity is a measure of the amount of data that it must process in order to be efficient.

There are certain key characteristics that an algorithm must possess. These characteristics are:

1. An algorithm must comprise of a number of steps.
2. It should have zero or more clearly defined input values.
3. It should be able to generate at least a single valid output based on a valid input.
4. It must be definite
5. It should be correct, i.e., it should be able to perform the desired task of generating correct output
from the given input.
6. There should be no ambiguity regarding the order of execution of algorithm steps.
7. It should be able to terminate by its own.

Example of Algorithm:

Algorithm to interchange two numbers:

```
Interchange (X, Y)
Step 1: Begin
Step 2: Set X = X + Y
Step 3: Set Y = X - Y
Step 4: Set X = X - Y
Step 5: Write (X, Y)
Step 6: End
```

Q.6. Elaborate data structure operations in detail.

Ans:

Data Structure is the way of organizing and storing data in a computer system so that it can be used efficiently. Data structures help in storing,organizing, and analyzing the data in a logical manner. The following points highlight the need of data structures in computer science:

1. It depicts the logical representation of data in computer memory.
2. It represents the logical relationship between the various data elements.
3. It helps in efficient manipulation of stored data elements.
4. It allows the program to process the data in an efficient manner.

There are several common operations associated with data structures that are used for manipulating the stored data. While defining a data structure, you also need to define these associated operations. The following operations are most frequently performed on any data structure type:

1. Traversing : It is the process of accessing each record of a data structure exactly once. Traversing a Data Structure means to visit the element stored in it. It visits data in a systematic manner.

2. Searching : It is the process of identifying the location of record that contains specific key value. Searching means to find a particular element in the given data-structure. It is considered as successful when the required element is found. Searching is the operation which we can performed on data-structures like array, linked-list, tree, graph, etc.

3. Inserting : It is the process of adding a new record in to a data structure. It is the operation which we apply on all the data-structures. Insertion means to add an element in the given data structure. The operation of insertion is successful when the required element is added to the required data-structure. It is unsuccessful in some cases when the size of the data structure is full and when there is no space in the data-structure to add any additional element. The insertion has the same name as an insertion in the data-structure as an array, linked-list, graph, tree. In stack, this operation is called Push. In the queue, this operation is called Enqueue.

4. Deleting : It is the process of removing an existing record from a data structure. It is the operation which we apply on all the data-structures. Deletion means to delete an element in the given data structure. The operation of deletion is successful when the required element is deleted from the data structure. The deletion has the same name as a deletion in the data-structure as an array, linked-list, graph, tree, etc. In stack, this operation is called Pop. In Queue this operation is called Dequeue.

Apart from these typical data structure operations, there are some other operations associated with data structures, such as-

1. **Sorting :** It is the process of arranging the records of data structure in specific order such as alphabetical, ascending, or descending. Sorting data in a particular order (ascending or descending).
We can take the help of many sorting algorithms to sort data in less time. Example: bubble sort which takes O(n^2)time to sort data. There are many algorithms present like merge sort, insertion sort, selection sort, quick sort, etc.
2. **Merging :** It is the process of combining the records of two different sorted data structures to produce a single sorted data set.Merging data of two different orders in a specific order may ascend or descend. We use merge sort to merge sort data.

Q.7. Explain asymptotic notations.

Ans:

Asymptotic notation is the most simple and easiest way of describing the running time of an algorithm. It represents the efficiency and performance of an algorithm in a systematic and meaningful manner. Asymptotic notations describe time complexity in terms of three common measures, best case (or 'fastest possible'), worst case (or 'slowest possible'), and average case (or 'average time').

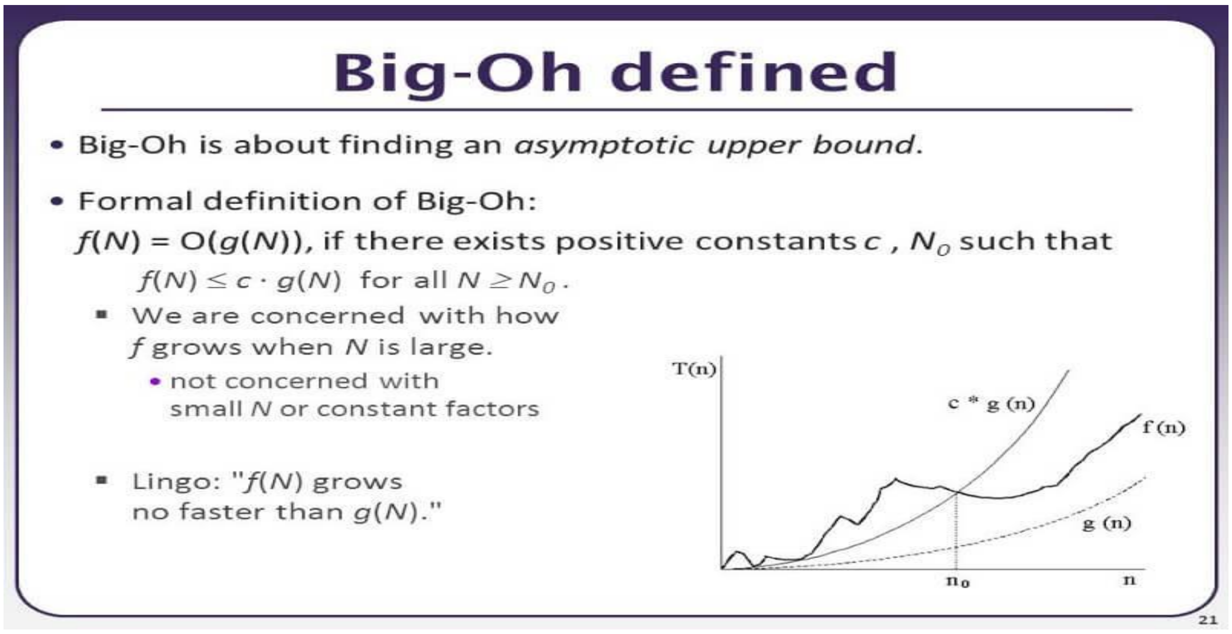
The three most important asymptotic notations are:

1. Big-Oh notation
2. Omega notation
3. Theta notation

1. Big-Oh Notation

The big-oh notation is a method that is used to express the upper bound of the running time of an algorithm. It is denoted by 'O'. Using this notation, we can compute the maximum possible amount of time that an algorithm will take for its completion.

Definition:



Big-Oh defined

- Big-Oh is about finding an *asymptotic upper bound*.
- Formal definition of Big-Oh:
  $f(N) = O(g(N))$, if there exists positive constants c , N_0 such that
  $f(N) \leq c \cdot g(N)$ for all $N \geq N_0$.
  - We are concerned with how f grows when N is large.
    - not concerned with small N or constant factors
  - Lingo: "$f(N)$ grows no faster than $g(N)$."

T(n)
c * g (n)
f (n)
g (n)
n₀
n
21

Figure shows the graphical representation of big-oh notation.

Some of the typical complexities (computing time) represented by big-oh notation are:

1. O(1) :- Constant
2. O(n) :- Linear
3. O(n2) :- Quadratic
4. O(n3) :- C ubic
5. O(2n) :- Exponential
6. O(logn) :- Logarithmic

2. Omega Notation

The omega notation is a method that is used to express the lower bound of the running time of an algorithm. Omega notation is denoted by Ω. Using this notation you can complete the minimum amount of time that an algorithm will take for its completion.

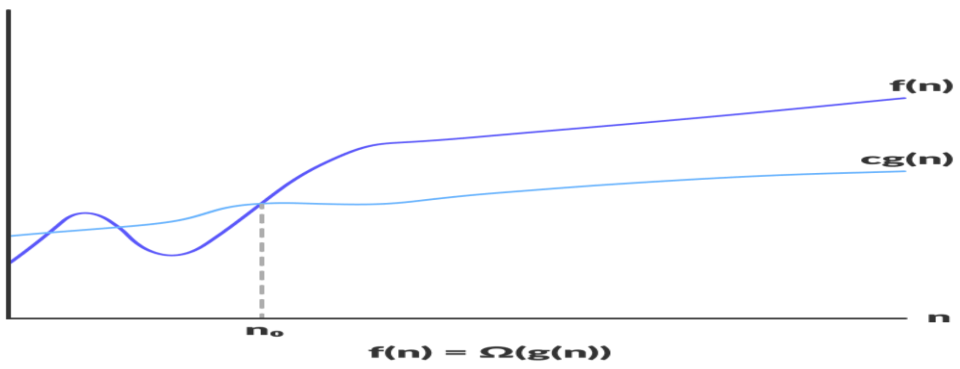


Figure: Graphical representation of Omega notation

3. Theta Notation

The theta notation is a method that is used to express the running time of an algorithm between the lower and upper bound. The theta notation is denoted by θ. Using this notation, we can compute the average time that an algorithm will take for its completion.

Let g and f be the function from the set of natural numbers to itself. The function f is said to be Θ(g), if there are constants c1, c2 > 0 and a natural number n0 such that c1* g(n) ≤ f(n) ≤ c2 * g(n) for all n ≥ n0

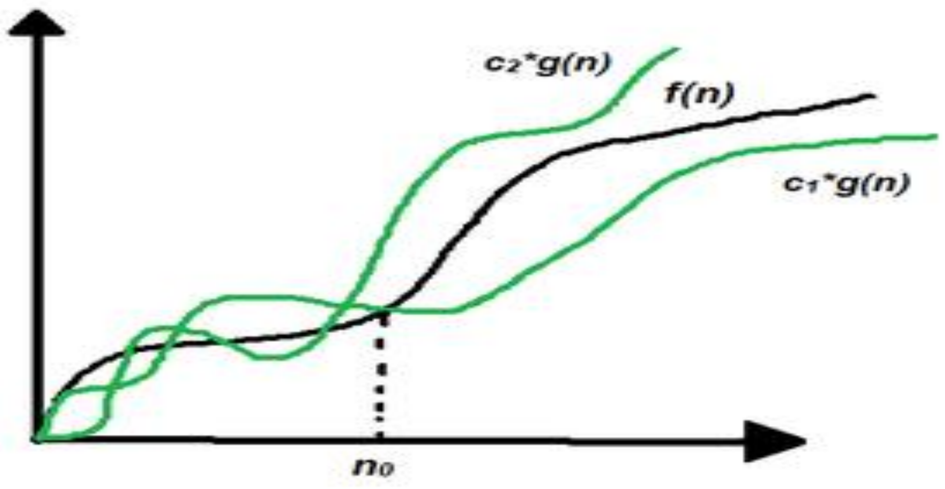


Figure: Graphical representation of Theta notation