Q. What do you understand by data structure? What are it’s importance? List out it’s application areas.

Ans: Data structure is a way of organizing data in a computer so that it can be accessed and used efficiently. It is a way of structuring data so that a computer program can effectively use it. Data structures are used to store, organize, and access data in an efficient manner. The importance of data structures lies in the fact that they allow for efficient operations on large amounts of data. Data structures are used to store data in a way that makes it easier to search, sort, and manipulate. They provide a convenient way of organizing data and can make data processing faster and easier. Data structures are also useful for performing complex calculations and for representing complex relationships between data elements. Data structures are important for a variety of reasons, including:

1. Improving efficiency of data operations: Data structures help to organize data in a way that makes it easier to search, sort, and manipulate. This can make data processing faster and easier.

2. Simplifying complex calculations: Data structures can be used to represent complex relationships between data elements, making complex calculations easier to perform.

3. Improving memory management: Data structures can help to manage memory more efficiently, helping to reduce the amount of memory needed to store data.

4. Making data more accessible: Data structures can make data more accessible, making it easier to find and use the data.

The application areas of data structure include:

1. Database Management System

2. Operating System

3. Compiler Design

4. Computer Graphics 5. Artificial Intelligence

6. Networking

7. Web Development

8. Data Mining

9. Natural Language Processing

10. Cryptography

Q. Difference between primitive and non-primitive data structures:

Primitive Data Structures	Non-Primitive Data Structures
Primitive Data Structure are predefined in the language.	Non-Primitive Data structure are not defined in language and created by the programmer.
Primitive Data structures will have a certain value.	Non-Primitive Data structure can have NULL value.
Primitive Data structures will have a certain value.	The size of non-primitive data structure are not fixed.
The primitive data structure starts with lowercase.	The non-primitive data type starts with an uppercase.
Can be used to call methods to perform operations.	Cannot be used.

Difference between linear and non-linear data structure:

Basis of	Linear Data Structure	Non-Linear Data Structure
Data arrangements	Data arrangement is in a linear sequence.	Data arrangement is not sequencing.
Transferring data Elements	It is possible to traverse the data elements in a run.	It is not possible to traverse the data elements in a run.
Implementation	Linear data structures are easier to implement.	Non-linear data structures are not easier to implement.
Levels	Single-level	Multiple level
Time complexity	Time complexity(time) taken to perform operations) changes as the data size increases.	Time complexity remains exactly the same.
Examples	Array, Stack, Queue, Linked List, etc.	Tree and Graph

Q. What are the various operations that can be performed in data structures?

The various operations that can ve performed in data structures are :

1. Insertion: Adding a new data item into a data structure.

2. Deletion: Removing an existing data item from a data structure.

3. Traversal: Moving through the data structure and processing each data item.

4. Searching: Locating a specific data item in a data structure.

5. Sorting: Arranging the data items in a particular order.

6. Merging: Combining two data structures into one.

7. Splitting: Separating one data structure into two or more.

8. Reversal: Reversing the order of the data items in a data structure.

9. Hashing: Mapping data items to particular locations in a data structure.

Q.What is ADT?How it differs from data type?what are benefits of using ADT?

ADT stands for Abstract Data Type. It is a type of data structure that is defined by its behavior, rather than its implementation. ADTs are considered abstract because the implementation details of the data structure are not specified.

A data type is a specific kind of data structure that is defined by its storage format and its operations. For example, an integer data type is defined by its storage format (typically a 32-bit number) and its operations (such as addition, subtraction, multiplication, etc.).

The benefits of using ADTs are that they allow for more flexibility when designing data structures. Because the implementation details are not specified, different implementations of the same ADT can be used depending on the context. Additionally, ADTs can be used to encapsulate data and operations, providing a layer of abstraction that makes it easier to understand how a data structure works. Finally, ADTs can be used to create modular code, making it easier to debug and maintain.

Q. What is algorithm? What are the characteristics of good algorithm?

Algorithm is a set of instructions or steps used to solve a problem.

Characteristics of a good algorithm include:

1. Correctness: The algorithm should produce correct results for all valid inputs.
2. Finiteness: The algorithm should terminate after a finite number of steps.
3. Input: The algorithm should have 0 or more well-defined inputs.
4. Output: The algorithm should produce 1 or more well-defined outputs.
5. Generality: The algorithm should be applicable to a wide variety of problems.
6. Efficiency: The algorithm should be efficient in terms of time and memory complexity.
7. Clarity: The algorithm should be clearly written and concise.

Q.What are asymptotic notation? Explain it's types.

Asymptotic notation is a way of expressing the complexity of an algorithm. It typically describes the run time or space requirements of an algorithm in terms of how the run time or space requirement grows as the input size grows. There are three main types of asymptotic notation:

Big-O notation, Omega notation, and Theta notation. Big-O notation: Big-O notation is used to describe the upper bound of an algorithm's complexity. It is typically used to describe the worst case run time complexity of an algorithm. For example, an algorithm that has a Big-O notation of O(n2) means that the worst case run time complexity is proportional to the square of the size of the input.

Omega notation: Omega notation is the opposite of Big-O notation. It is used to describe the lower bound of an algorithm's complexity. It is typically used to describe the best case run time complexity of an algorithm. For example, an algorithm that has an Omega notation of Ω(n) means that the best case run time complexity is proportional to the size of the input.

Theta notation: Theta notation is used to describe the exact complexity of an algorithm. It is used to describe the average case run time complexity of an algorithm. For example, an algorithm that has a Theta notation of Θ(n2) means that the average case run time complexity is proportional to the square of the size of the input.

Q.ADT supports extraction. Explain.

ADT stands for Abstract Data Type, and it is a type of data structure used to store and organize data in a certain way. ADT supports extraction because it allows developers to extract data from the data structure and store it in another location. This extraction process can be done by using various methods, such as sorting, searching, and traversing. By using these methods, developers can easily access and manipulate the data stored in ADT.

Q.Point out the limitation and advantages of different asymptotic notations.

Limitations:

1. Asymptotic notation does not provide precise bounds, only an indication of the order of growth of a function.

2. Asymptotic notation is not always a good indicator of the actual performance of a program.

Advantages:

1. Asymptotic notation provides a useful tool for comparing the relative efficiency of algorithms.

2. Asymptotic notation can help determine the best algorithm to use in a given situation.

3. Asymptotic notation can be used to compare algorithms on a theoretical level, without the need to actually implement and compare them.

Q. How can you say stack is ADT? Explain.

Stack is a linear data structure in which the insertion and deletion operations are performed at only one end. In a stack, adding and removing of elements are performed at a single position which is known as "top". That means, a new element is added at top of the stack and an element is removed from the top of the stack. In stack, the insertion and deletion operations are performed based on LIFO (Last In First Out) principle. Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.

Stack Representation

In a stack, the insertion operation is performed using a function called "push" and deletion operation is performed using a function called "pop".

Stack ADT

A Stack contains elements of the same type arranged in sequential order. All operations take place at a single end that is top of the stack and following operations can be performed:

1. push() – Insert an element at one end of the stack called top.
2. pop() – Remove and return the element at the top of the stack, if it is not empty.
3. peek() – Return the element at the top of the stack without removing it, if the stack is not empty.
4. size() – Return the number of elements in the stack.
5. isEmpty() – Return true if the stack is empty, otherwise return false.
6. isFull() – Return true if the stack is full, otherwise return false.