1. What are Data Structures?

* Data structures are ways to organize, store, and manage data efficiently in a computer so that it can be used effectively.
* Creating data structures is important because they help manage and organize data in a way that makes it easy to perform tasks efficiently. Without data structures, working with data would be slow, unorganized, and prone to errors.

1. Describe the types of Data Structures?

* Linear Data Structure: A data structure that includes data elements arranged sequentially or linearly, where each element is connected to its previous and next nearest elements, is referred to as a linear data structure. Arrays and linked lists are two examples of linear data structures.
* Non-Linear Data Structure: Non-linear data structures are data structures in which data elements are not arranged linearly or sequentially. We cannot walk through all elements in one pass in a non-linear data structure, as in a linear data structure. Trees and graphs are two examples of non-linear data structures.

1. What is a stack data structure? What are the applications of stack?

* A stack is a type of data structure that works on the principle of LIFO (Last In, First Out). This means: The last item added (pushed) to the stack is the first one to be removed (popped).
* Push: Add an item to the stack (like putting a plate on top).
* Pop: Remove an item from the stack (like taking the top plate off).
* Peek: Look at the top item without removing it.
* IsEmpty: Check if the stack is empty.
* Applications: Reversing a string, Parenthesis matching.

1. What is a queue data structure? What are the applications of queue?

* A queue is a type of data structure that follows the First In, First Out (FIFO) rule. This means the first element added to the queue will be the first one to be removed.
* Items are added from one end, called the rear (or back), Items are removed from the other end, called the front.
* enqueue: This adds an element to the rear end of the queue. Overflow conditions occur if the queue is full.
* dequeue: This removes an element from the front end of the queue. Underflow conditions occur if the queue is empty.
* isEmpty: This returns true if the queue is empty or else false.
* rear: This returns the rear end element without removing it.
* front: This returns the front-end element without removing it.
* size: This returns the size of the queue.
* Applications: Printing Jobs, Task Scheduling.

1. Differentiate between stack and queue data structure.

* A stack is a data structure that follows the Last In, First Out (LIFO) rule. The last item added is the first one to be removed. A queue is a data structure that follows the First In, First Out (FIFO) rule. The first item added is the first one to be removed.
* In stack, Items are added at the top (end) and are removed from the **top** (end). In queue, Items are added at the rear (back) and are removed from the front (start).

1. What is array data structure? What are the applications of arrays?

* An array is a collection of items (data) stored in a continuous block of memory. All items in an array are of the same type (e.g., all integers or all strings), and each item can be accessed using its index (position in the array).Indexing starts from 0, meaning the first item is at index 0, the second at index 1, and so on. Arrays are used to store multiple values in a single variable.
* Storing Data: Arrays are used to store a list of items like names, numbers, or any other data that needs to be accessed quickly.
* Storing Images: Arrays store pixel data for images, where each element represents a color value.

1. Elaborate on different types of array data structure.

* One-dimensional array: A one-dimensional array stores its elements in contiguous memory locations, accessing them using a single index value. It is a linear data structure holding all the elements in a sequence.
* Two-dimensional array: A two-dimensional array is a tabular array that includes rows and columns and stores data. An M × N two-dimensional array is created by grouping M rows and N columns into N columns and rows.

1. What is a linked list data structure? What are the applications for the Linked list?

* A linked list is a data structure made up of nodes, where each node contains:
* Data: The value or information stored in the node.
* Pointer: A reference or link to the next node in the list.
* Unlike arrays, the elements in a linked list are not stored in continuous memory locations, they are stored randomly. Instead, each node points to the next one, forming a chain.
* Singly Linked List: Each node points to the next node.
* Doubly Linked List: Each node has links to both the next and the previous nodes.
* Circular Linked List: The last node points back to the first node.
* Applications: Implementation of Other Data Structures, Music Playlist.

1. Difference between Array and Linked List.

* An array is a collection of elements stored in continuous memory locations. A linked list is a collection of nodes where each node points to the next (and sometimes previous) node.
* Array has Fixed size; requires contiguous memory. Linked list has Dynamic size; elements are stored in scattered memory locations and can be added in any location.
* Array is Fast due to direct access using index. Linked list is Slower because sequential access is required.

1. What is an asymptotic analysis of an algorithm?

* Asymptotic Analysis is a way to evaluate the performance of an algorithm, especially its time complexity and space complexity, as the size of the input data grows very large. It helps us understand how efficiently an algorithm performs when handling large inputs.
* Three Notations: Big-O Notation: which describes the **upper bound** of an algorithm’s runtime or space usage. Omega (Ω): Describes the best-case scenario. Theta (Θ): Describes the average-case scenario or the tight bound.

1. What is hashmap in data structure?

* A HashMap is a data structure that stores key-value pairs, where each key is unique, and it is used to quickly find, insert, and delete data based on the key.
* Hashing: When a key is added to a HashMap, a special function called a hash function converts the key into a number (called a hash code). This hash code determines where the value associated with the key will be stored in memory.
* Buckets: The hash code maps the key-value pair to a specific location (bucket) in the HashMap.
* Collisions: Sometimes, different keys can produce the same hash code. This is called a collision. HashMaps handle collisions using methods like chaining (storing multiple values in a single bucket using a list) or open addressing (finding another free spot).
* You can access data in O(1) time on average.

1. What is a priority queue? What are the applications for priority queue?

* A Priority Queue is a special type of queue where each element has a priority. Instead of processing elements in the order they arrive (like a regular queue), the element with the highest priority is processed first. If two elements have the same priority, they are processed according to their order of arrival.
* Each element has a value and a priority. The queue is arranged based on priority. High-priority elements come to the front. Lower-priority elements stay at the back.

1. What is graph data structure? What are the applications for graphs?

* A graph is a data structure used to represent relationships between different items. It consists of: Nodes (or Vertices): These are the items (like people, cities, or devices). Edges: These are the connections or relationships between nodes (like friendships, roads, or network links).
* Graphs can be: Directed: The edges have directions (e.g., one-way streets). Undirected: The edges do not have directions (e.g., two-way roads).
* Applications: Maps and Navigation, Web Page Links, Recommendation Systems.

1. What is Tree data structure? What are the applications for Trees?

* A Tree is a type of data structure that looks like an upside-down tree, where:
* Nodes store data. Each node is connected to other nodes through branches or edges. It starts with a root node (the topmost node) and expands into multiple child nodes. Nodes that have no children are called leaf nodes.
* Applications: File Systems, Decision-Making in machine Learning (Decision tree).

1. Explain the concept of a binary search tree.

* A Binary Search Tree (BST) is a special type of tree data structure where:
* Each node has at most two children: left and right.
* Left child contains values smaller than the parent node.
* Right child contains values greater than the parent node

1. What is the concept of the bubble sort algorithm?

* Bubble Sort is a simple sorting algorithm that repeatedly steps through a list of elements, compares adjacent items, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.
* Compare the first two elements of the list.
* If the first element is greater than the second, swap them.
* Move to the next pair of adjacent elements, and repeat the process.
* Continue this until the largest element "bubbles" up to the end of the list.
* Repeat the process for the rest of the list (ignoring the last sorted element).
* Continue until no swaps are needed, which means the list is sorted.

1. Can you explain string hashing?

* String Hashing is a technique used to convert a string (a sequence of characters) into a fixed-size number (called a hash value or hash code). This number represents the string in a compressed format, making it easier and faster to compare strings or search for them.
* Assign a value to each character: For example, in the ASCII encoding, the character 'A' has the value 65, 'B' is 66, and so on.
* Combine the values: The individual values of characters are combined using mathematical operations like addition, multiplication, or modular arithmetic. This results in a unique number (hash) for the string.
* Modular operation: The result is often reduced using a modular operation (like dividing by a prime number) to keep the hash within a fixed size.

1. Explain Sequential sort.

* The main idea behind sequential sort is to look at every element in the list and find where it should be placed in the sorted order. Here’s how it works step-by-step:
* Start at the beginning: Begin with the first element in the list.
* Compare each element: Compare the current element with the next element in the list.
* Swap if needed: If the current element is greater than the next element (for ascending order), swap them. Otherwise, leave them as they are.
* Repeat: Move to the next element and repeat the process until the entire list is sorted.
* Finish when sorted: Keep doing this until no more swaps are needed, meaning the list is sorted.

1. Explain Insertion sort.

* Start with the second element: In the beginning, the first element of the list is already considered sorted by itself. You then take the second element and compare it to the first element.
* Compare and Insert: If the second element is smaller than the first, you swap them. Now, the first two elements are sorted.
* Move to the next element: You move to the third element. Compare it with the elements before it (from right to left) and insert it in the correct position.
* Repeat for all elements: Continue this process for all elements, inserting each element into its correct position among the previously sorted elements.

1. Explain Quick sort.

* Quick Sort is a fast and efficient sorting algorithm used to arrange a list of items (like numbers or words) in a specific order, typically in ascending or descending order. It follows a divide-and-conquer approach, meaning it divides the problem into smaller parts and solves each part individually.
* Choose a pivot element: A pivot is selected from the list (usually the first, last, or middle element). This pivot will be used to divide the list into two parts: one part with elements smaller than the pivot and the other with elements larger than the pivot.
* Partition the list: The list is rearranged so that all elements less than the pivot are on one side, and all elements greater than the pivot are on the other side. The pivot is now in its correct position in the sorted list.
* Recursively sort the sublists: After the pivot is placed in the correct position, the list is divided into two smaller lists (one on the left of the pivot and one on the right). These smaller lists are then sorted using the same process until the entire list is sorted.

1. Explain Merge sort.

* Merge Sort is a popular sorting algorithm that follows the Divide and Conquer approach to sort a list or an array. It divides the list into smaller parts, sorts those parts, and then merges them back together in order.
* Divide:First, the algorithm divides the unsorted list into two halves.
* Recursively Sort: Each half is sorted by repeatedly dividing it into two halves until each sublist has only one element. A list with one element is already sorted.
* Merge: Once the sublists are sorted, the algorithm merges them back together in the correct order. During the merge step, it compares the elements of both sublists and combines them in order, one element at a time.

1. Explain heap sort.

* Heap Sort is a sorting algorithm that uses a special data structure called a heap to sort elements. A heap is a binary tree where the parent node is always greater than (in a max-heap) or smaller than (in a min-heap) its child nodes.
* Heap Sort works by first arranging the elements into a heap and then repeatedly extracting the maximum (or minimum) element to get a sorted list.
* Build a Heap:
  + Start by converting the input list (or array) into a heap.
  + Max-Heap: The largest element is at the top (root).
  + Min-Heap: The smallest element is at the top (root).
  + This step ensures that the heap property is satisfied for every node in the tree.
* Extract Elements from the Heap:
  + Once the heap is built, the largest element (in a max-heap) or the smallest element (in a min-heap) is always at the root.
  + Swap the root element with the last element in the heap (which will be in its final sorted position).
  + Reduce the size of the heap by excluding the last element (now in its correct position).
  + Heapify the remaining heap to restore the heap property.
  + Repeat the process until all elements are sorted.

1. Explain Time complexity and space complexity.

* How much time does the algorithm take to complete the task? It helps us understand if the algorithm will run fast or slow when we give it a larger set of data.
* How much memory does the algorithm require to store data during execution? It helps us understand if the algorithm uses a lot of memory or a little memory as the input grows.