DATA STRUCTURE WEEK - 1

- **Data Structure** - A data structure is a way of organizing and storing data in a computer so that it can be accessed and used efficiently. It refers to the way in which data is organized, managed, and stored in a computer's memory. Data structures provide a way of representing complex data in a more organized and manageable way, making it easier to access and manipulate.
  Efficient use of data structures can greatly improve the performance of computer programs, as they allow for faster data access and manipulation.

- **Algorithm** - An algorithm refers to a step-by-step procedure used to solve a problem or accomplish a task, where the problem and the input are represented by some data structure.

- **Linear Data Structure** - A linear data structure is one in which the data elements are arranged in a sequential order, where each element is connected to its previous and/or next element. Examples of linear data structures include arrays, linked lists, stacks, and queues. Traversal of linear data structures can be done in a linear fashion, from one end to the other.

- **Non-Linear Data Structure** - A non-linear data structure is one in which the data elements are not arranged in sequential order, and the elements may be connected in an arbitrary manner. Examples of non-linear data structures include trees, graphs, and heaps. Traversal of non-linear data structures can be more complex and require specialized algorithms, such as breadth-first or depth-first searches.

- **Complexity analysis** is an essential part of designing and analyzing algorithms in Data Structures and Algorithms (DSA). It helps to understand the performance of an algorithm concerning the input size, which is critical in determining the algorithm's efficiency.

    1. **Time complexity analysis**: It refers to the amount of time an algorithm takes to execute concerning the input size. Time complexity is usually denoted by the big O notation.
    O(1) constant time: The algorithm's runtime is constant, and it doesn't depend on the input size.
    O(n) linear time: The algorithm's runtime increases linearly with the input size.

O(n^2) quadratic time: The algorithm's runtime increases as the square of the input size.
O(log n) logarithmic time: The algorithm's runtime increases logarithmically with the input size.
O(n log n) log-linear time: The algorithm's runtime increases as n times the logarithm of n.
O(2^n) exponential time: The algorithm's runtime increases exponentially with the input size.

2. **Space complexity analysis**: It refers to the amount of memory an algorithm uses concerning the input size. Space complexity is also denoted by the big O notation.
O(1) constant space: The algorithm uses a fixed amount of memory, and it doesn't depend on the input size.
O(n) linear space: The algorithm's memory usage increases linearly with the input size.
O(n^2) quadratic space: The algorithm's memory usage increases as the square of the input size.

- **Contiguous** memory allocation is used in many data structures such as **arrays** and matrices. For example, an array in most programming languages is a contiguous block of memory where each element of the array is stored in a consecutive memory location.

- **Noncontiguous** memory allocation is an important technique in data structures, enabling more efficient use of memory and allowing for more flexible memory allocation and management. Ex - **Linked List**

- **Asymptotic Notation -** are the expressions that are used to represent the complexity of an algorithm
    - **Worst Case, Best Case, Average Case**
    - **Big O notation** is a way of describing the time complexity of an algorithm, which is a measure of how much time an algorithm takes to run as a function of the size of its input. Big O notation specifically describes the **worst-case** scenario.
    - **Omega Notation** - Omega notation specifically describes the **best-case** scenario.

    - **Theta Notation -** Theta notation specifically describes the **average** case scenario.

- **Memory leaks** - Memory leaks occur when a program allocates memory but fails to free it when it is no longer needed. This can lead to memory exhaustion and eventually cause the program to crash or hang.

- **ARRAY**
    - Init - O(n)ST
    - Set - O(1)ST
    - Get - O(1)ST
    - Traverse - O(1)S, O(n)T
    - Insert - O(1)S, O(n)T
    - Delete - O(1)S, O(n)T **(if deleting at the end of the array - O(1)S, O(1)T)**
    - Accessing an element in an array takes constant time O(1), while searching for an element in an unsorted array takes linear time O(n). Insertion and deletion operations can take up to linear time O(n) in the worst case, depending on the position of the element in the array.

- **DYNAMIC ARRAY -** The dynamic array is initially allocated with a fixed size, and elements are added or removed from the array as needed. If the array becomes full, a new, larger array is allocated and the elements of the old array are copied into the new array. This process is known as resizing the array, and it allows the array to grow in size when needed. - O(1)ST, in the worst case (when an array is full and reallocating to a new array) - O(n)ST

- **LINKED LIST -**
    - Init - O(1)ST
    - Set - O(1)S, O(n)T
    - Get - O(1)S, O(n)T
    - Traverse - O(1)S, O(n)T
    - Insert - At the Beginning(**HEAD**) - O(1)ST, At Ending(**TAIL**) - O(1)ST, at some cases when the variable is not in the tail - O(n)T, O(1)S
    - Delete - O(n)ST, At **Beginning** And **End** O(1)ST
        - **SINGLE LINKED LIST** - A singly linked list is a data structure used to store a sequence of elements, called nodes. Each node in a singly linked list contains a value or data and a reference or a link,

to the next node in the list. The last node in the list typically has a null reference to indicate the end of the list. **Accessing a specific node in the list can be slower than in an array.**

- **DOUBLE LINKED LIST -** Each node in a doubly linked list contains two pointers or references - one that points to the previous node and one that points to the next node. This allows for efficient traversal in both directions, making it easier to insert, delete, or search for elements in the list.

- **CIRCULAR LINKED LIST** - A circular linked list is a data structure in which each node contains a data value and a pointer to the next node in the list. However, in a circular linked list, the last node's pointer points back to the first node in the list instead of being null, forming a circle.
  - **APPLICATIONS**
    - **Music and video streaming services**: Linked lists are used to maintain playlists and track user preferences.
    - **Network routing**: Linked lists can be used to represent the topology of a network and to route data between nodes in the network.
    - **Web browsing**: Linked lists are used to represent the history of web pages visited by a user and to implement the "back" and "forward" buttons in a web browser.

- **Linear Search -** Linear Search is used to find a specific element in an array or a list of elements. It sequentially searches through each element of the array or list until it finds the target element or reaches the end of the list. Linear search is a basic algorithm with a time complexity of **O(n)**. It is not very efficient for large datasets, but it is helpful for small datasets or when the data is unsorted or the search space is limited.

- **Binary Search** - Binary search is an efficient algorithm used for searching an element in a sorted array or list of elements. It works by repeatedly dividing the search interval in half until the target element is found or the search interval is empty. Binary search has a time the complexity of **O(log n)**, This makes it much faster than linear search, especially for large datasets. **it only works on sorted**

datasets and requires extra preprocessing time to sort the data before searching.

- **RECURSION** - Recursion is a programming technique where a function calls itself one or more times. Essentially, a recursive function is one that solves a problem by breaking it down into smaller, simpler versions of the same problem until a base case is reached that can be solved directly.
    - **ADVANTAGES**
        - Recursive functions can be easier to read and understand since they break down a problem into smaller, more manageable pieces.
        - Recursive functions can provide an elegant and concise solution to complex problems that would be difficult to express using iterative methods.
    - **DISADVANTAGES**
        - Recursive functions can be less efficient than iterative methods because they typically require more memory and processing power to execute.
        - If it is not implemented properly then it is a very slow process. And it is a very difficult task to write a recursive function with much less speed and low memory.
- **Mutable vs immutable string -** Mutable is a string that can be changed after it has been created. An immutable string is a string that cannot be changed once it has been created.
- **Deallocation** - Deallocation refers to the process of freeing up memory that has been previously allocated for use by a computer program. When a program requests memory from the operating system, the operating system reserves a block of memory and assigns it to the program. When the program is finished using the memory, it needs to release it so that it can be used by other programs. Failing to properly de-allocate memory can result in memory leaks.