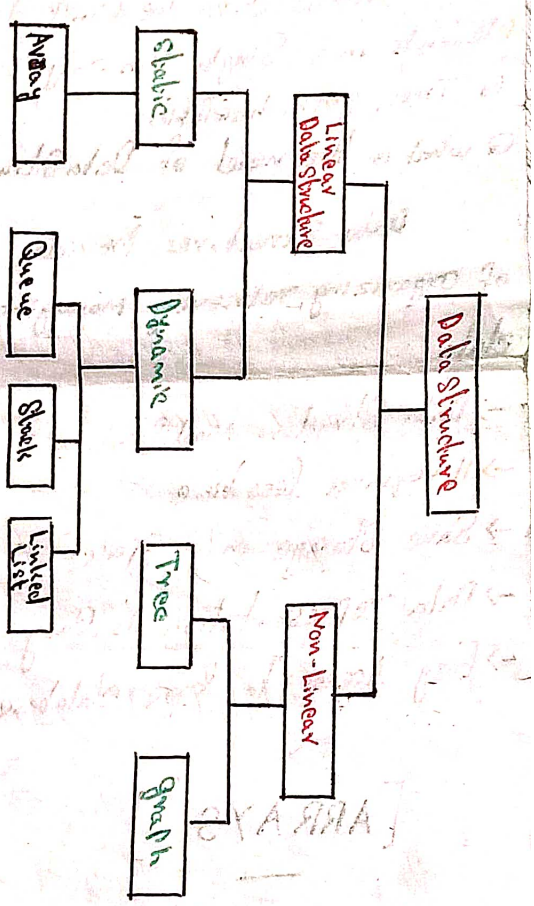
DATA STRUCTURES

Q what is Data Structure ?

A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently.

★ Data Structure is not only for organizing it is for processing, it is for retrieving and storing data.



=> Types of Data Structures

★ Linear Data Structure : Data Structures which data elements are arranged sequentially or linearly, where each element is attached to it's previous and next adjacent elements.

● -> Types of Linear Datastructure

• Static Data Structure : Static data structure has fixed memory size. It is easier to access the elements in a Static Data Structure.

Eg: Array

• Dynamic datastructure: in dynamic data structure the size is not fixed. It can be randomly updated during run time.

Eg: queue, stack

★ Non-Linear DataStructure

In Non-linear DataStructure the elements are not placed sequential or linearly is called non-linear data structure. In non-linear datastructure we can't traverse all the elements in a single run only.

Eg: Trees, graph, hashTables.

Q what is the need of DataStructure?

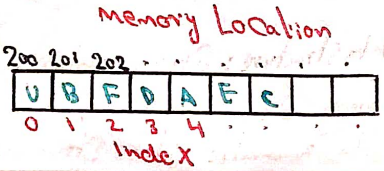
Data Structures provides an easy way of organizing, retrieving, managing and storing data

-> Data Structure modification is easy
-> It requires less time
-> Save Storage m/y space
-> Data representation is easy
-> Easy Access to large database.

[ARRAYS]

Q what is the Concept of an Array?

An array is a linear datastructure, and it is a collection of items stored at Contigous memory locations. In other words we can say that an array is a collection of same type of elements.



=> Characteristics of Array.

1) Array is index based. So identifying each element is easier
2) It stores multiple values of same type
3) It can handle complex data structures by two dimensional array.
4) Search process on array is easy.

=> Applications of Array

1) Arrays used in solving matrix problems
2) Database records are also implemented by array
3) It is used to implement other data structures like Stack, Queue, Heaps and Hash tables.
4) An array can used for CPU scheduling

=> Real-Life Applications of Array.

1) Used for Computing Mathematical Operations
2) Used for image Processing
3) Used for Record Management
4) Used in ordering Boxes.

ALGORITHM

Q what is an Algorithm ?

An Algorithm is a step by step procedures for solving a problem or performing a task. It is a finite set of instructions that can be executed by a computer to accomplish a particular goal.

-> In data structure, algorithms are used to manipulate and organize data in an efficient and effective manner.

-> The common Algorithms used in data structures are sorting algorithms, searching algorithms, traversal algorithms. The efficiency of an algorithm is often measured in terms of its time and space complexity.

Q Algorithm to Print 0-N

```
void main()
{
Print("Enter the value");
Var N = int.Parse(stdin.readLineSync()!);
var i=0;
while (i<=N)
{
Print(i);
i++;
}
}
```

=> Algorithm Steps

1) Start
2) Read the value of N from the User
3) Set the value of i to 0
4) while i is less than or equal to N, do the following
   a) Print the value of i Using Print function
   b) Increment the value of i by 1
5) End the while loop
6) End

=> Memory Allocation

Memory Allocation in Data Structures refers to the process of allocating and deallocating memory space for storing data in a Computer's memory, in other words, it is the process of reserving a block of memory for a Program or application.

In DataStructures Memory allocation is a Critical aspects that determines the efficiency and Performance of algorithms and Programs.

=> Two Types of memory Allocations

1) Static memory Allocation
2) Dynamic memory Allocation

-> Static Memory Allocation

In Static Memory Allocation, the memory for a Program or data structure is allocated at Compile-time and remains fixed throughout the Program.

-> Dynamic Memory Allocation

In Dynamic Memory Allocation, memory is allocated during the runtime.

=> Memory Leak

Memory leak is a situation where a Program or application uses memory but fails to release it when it is no longer needed overtime, this can result in a shortage of available, leading to reduced Performance instability, and even crashes.

-> Reasons for memory leak

- Failure to deallocate memory
- Circular references
- Unreleased resources

=> Complexity Analysis

Complexity analysis is the process of analyzing the performance of algorithms & datastructures in terms of time and space complexities. It helps us to understand how much time and memory an algorithm or data structure require.

=> Complexities of Common operation of all DataStructures

1) Arrays • Access : O(1) • Search : O(n) • Insertion : O(n) • Deletion : O(n)	2) Linked Lists • Access : O(n) • Search : O(n) • Insertion : O(1) • Deletion : O(1)
3) Stack • Push : O(1) • Pop : O(1) • Peek : O(1)	4) Queues • Enqueue : O(1) • Dequeue : O(1) • Peek : O(1)
5) Hash Tables • Search : O(1) • Insertion : O(1) • Deletion : O(1)	6) Trees • Search : O(log n) • Insertion : O(log n) • Deletion : O(log n)
7) Heap • Insertion : O(log n) • Deletion : O(log n) • Peek : O(1)	

Q what is Space Complexity ?

Space Complexity is the Amount of memory required by a Data Structure or an algorithm.

Q what is Time Complexity ?

Time Complexity is the Amount of Time required to Complete a Task

=> Asymptotic Analysis [Big-O notation]

Asymtotic Analysis is a way to measure the efficiency of algorithms, which are sets of instructions. When we Analyze the algorithms we are interested to understanding how it's Performance Changes as the Size of the Problem it Solves increases.

=> Use of Asymptotic Analysis

The goal of Asymptotic Analysis is to Compare the efficiency of different algorithms and to Choose the most efficient for the given Problem.

-> Usually time required by an algorithm falls under three types.

1) Best Case
2) Average Case
3) Worst Case

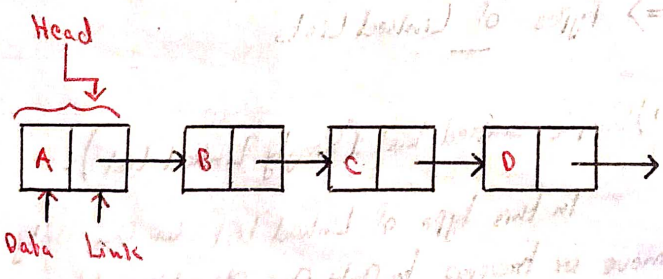
=> Asymptotic Notations

1) O - Big Oh Notation
2) Ω - Big Omega Notation
3) Θ - Theta Notation
4) o - Little Oh Notation
5) ω - Little Omega Notation

=> Linked List

Linked List is a Linear datastructure like arrays. In linked list elements are not stored at a contiguous location the elements are linked using pointers.

In linked list each contains a data part and link part (Address part).



Q Why we use the Linked List?

The Linked List can manage the data operations rather than array.

- The Size of Array is fixed: we know the upper limit of array. So allocated memory is equal to the upper limit.
- Insertion/Deletion is expensive: in array the insertion and deletion process is done by shifting of elements. But in Linked List we can traverse to any node. and can insert a new node in required position.

=> Advantages of Linked List

1) Dynamic Array
2) Ease of insertion/Deletion
3) insertion at begining takes O(1), where as array take O(n)

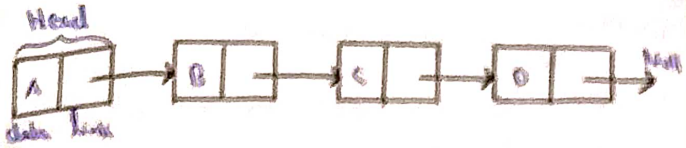
=> Drawback of Linked List

1) Random Access is not allowed, which means we have to start with initial node Head node. [binary search is not possible in linked list]
2) Extra memory for pointer is required
3) Not Cache-friendly. In array there is locality relevence, which is not in the case of linked list
4) Takes more time for traversing and changing pointers
5) Reverse traversing not possible.

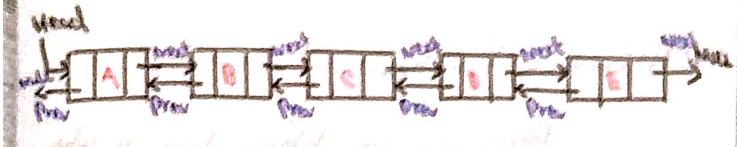
=> Types of Linked Lists

1) Simple Linked List (Singly Linked List):

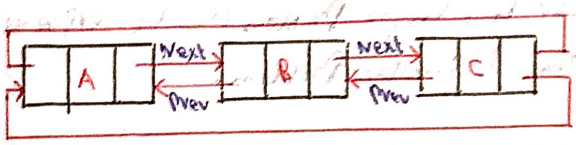
In this type of Linked List we can only move or traverse to only one direction. where each pointer of each nodes pointers to other node, where last node points to null.



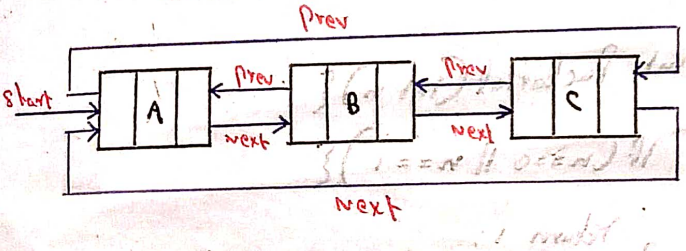
2) Doubly Linked List: In this type of Linked List we can move or traverse in both direction. (Forward and backward)



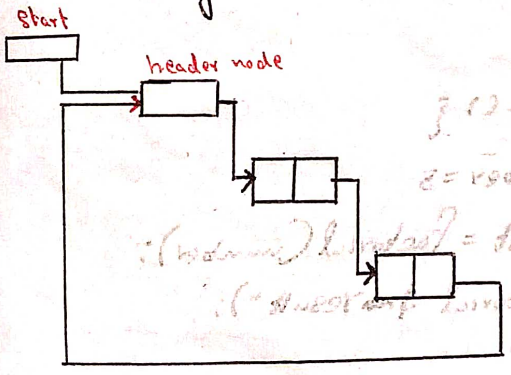
3) Circular Linked List: In this type the last node of the list points to the Head node/first node



4) Doubly Circular Linked list: It is the most complex type of linked list which contains pointer as well to next node as well as previous node in the sequence. The circular doubly linked list does not contain null in the previous field of the first node.



5) Header Linked List: A header linked list is a special type of linked list that contains a header node at the beginning of the list.



=> Applications of Linked List.

- Linked List can be used to represent useful data-structures like Stacks and queues
- Linked List can be used to implement hash tables each bucket of the hash table can be linked list.
- Linked List can be used to implement graphs

=> Recursive function

Recursion in datastructure is when a function calls itself directly or indirectly. The function calling itself is known as a recursive function. It is generally used when the answer to a larger problem can be expressed in terms of smaller problems.

Eg: def Print_list(list):
    if list is Empty:
        return
    Print list[0],
    Print_list(list[1:])

Eg: ~~Recursion~~ to factorial problem.

```
int factorial(int n){
  if(n==0 || n==1){
    return 1;
  }
  else{
    return n*factorial(n-1);
  }
}
void main(){
 int number=5
 int result = factorial(number);
Print("factorial %d", result");
}
```