|  |  |
| --- | --- |
| **Data Structure** | |
| 1. Primitive Data Structure:   Data structure that can hold a single value.  EX: int , char , float. | |
| 1. Non-Primitive Data Structure: | |
| * **Non-Linear Data Structure:** * The arrangement of data in a random manner. * One element is connected to “n” elements. * EX: * Graph * Tree | * **Linear Data Structure:** * The arrangement of data in a sequential manner. * One element is connected to only one another element in a linear form. * EX: * Static: Array. * Dynamic: Stack, Queue, Linkedlist. |

|  |  |
| --- | --- |
| 1. **Array** | |
| * **Disadvantages of Array**:  1. Array is homogenous متجانس   It means that elements that can be stored in it must be with similar datatype.   1. Static memory allocation.   It means that the size of array can’t be changed in runtime.   1. There will be wastage of memory.   If we store less number of elements than the declared size. | * Each Element of array is same datatype * Each element in array can accessed via Index. * Sorting and searching in array is easier. * Arrays are good for storing multiple values in a single variable. * Continuous memory allocation. * Traversing in array is very simple process, we just need to increment the base address of array. |

|  |  |
| --- | --- |
| 1. **Stack: (LIFO)** | |
|  | * Collection of objects accessed from one end (**top**) * Stack is empty 🡪 top=-1 * Stack is full 🡪top= maxSize * Stack Overflow 🡪When stack is full, and you try to push an element. * Stack underflow🡪When stack is empty, and you try to pop an element. |

|  |
| --- |
| 1. **Queue**: (FIFO) |
| * Types of Queue:  1. Linear Queue 2. Circular Queue      * + - Better memory utilization.   If the empty space is available in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear.   1. Priority Queue  * It is a special type of queue in which the elements are arranged based on the priority.      1. Deque (Double Ended Queue)  * Insertion and deletion can be done from both ends of the queue either from the front or rear. |

|  |  |
| --- | --- |
| 1. **Linkedlist** | |
| * **Disadvantages of Linkedlist:**  1. Memory usage.   In linkedlist, node have two variables.   1. Traversing.   The time required to access a particular node is large. | * **Why we use linkedlist over Array?** * The limitation of array:  1. Static memory allocation.   It is impossible to expand the size of array in run time.   1. Continuous memory allocation   So inserting an element in the array in particular position needs shifting many elements.   * Linkedlist is useful because:  1. Dynamic memory allocation. 2. Non continuous memory allocation. 3. Insertion & deletion is easier in linkedlist. 4. Memory efficient.   The size of linkedlist can grow & shrink at runtime. |
| * **Types of Linkedlist:**  1. **Singly Linkedlist**    * Only forward traversal is possible, we can’t traverse in the backward direction. 2. **Doubly linkedlist**      1. struct node 2. { 3. node \*prev; 4. **int** data; 5. node \*next; 6. } 7. **Circular singly linkedlist**     It is a singly linkedlist but the last node connects to the first node.   1. **Circular doubly linkedlist** | |