# DATA STRUCTURE WEEK - 2

- **Stack -** A stack is a linear data structure in which elements can be inserted and deleted only from one side of the list, called the **top**. A stack follows the **LIFO** (Last In First Out) principle, i.e., the element inserted at the last is the first element to come out. The insertion of an element into the stack is called **push** operation, and the deletion of an element from the stack is called **pop** operation. In stack, we always keep track of the last element present in the list with a pointer called **top**.
  For a dynamic stack implemented using a singly linked list, the time complexity of push and pop operations is O(1), which means that they take constant time  regardless of the size of the stack. For a static stack implemented using an array, the time complexity of push and pop operations is also O(1) as long as the stack is not full. However, if the stack is full, push operation will have a time complexity of O(n), where n is the size of the stack because the entire stack needs to be copied to a larger array.
  - **Types of stack**
    - **Static Stack**: A static stack is implemented using an array and has a fixed size. Once the stack is full, no more elements can be added to it.
    - **Dynamic Stack**: A dynamic stack is implemented using a linked list and can grow or shrink as needed.
      - Advantages
        - **Efficient memory usage**: Stacks are memory-efficient as they only store a limited number of elements at a time.
        - **Fast operation**: Stacks are fast as they only need to add or remove elements from one end
      - Disadvantages
        - **Limited storage capacity**: Static stacks have a fixed size, so if the stack is full, no more elements can be added to it.

        - **No random access**: Stacks do not allow for random access, which means that elements in the middle of the stack cannot be accessed directly.
        - **Stack overflow/underflow**: Stacks can suffer from overflow or underflow, which occurs when elements are added or removed beyond the stack's limits, leading to a program crash.
            - **Applications**
                - **Web browsers**: When you visit a website, the web browser uses a stack to keep track of the pages you've visited so that you can easily go back to the previous page by clicking the "back" button.
                - **Undo/redo functionality**: Many applications, such as text editors or graphic design software, use a stack to keep track of the user's actions, allowing them to undo or redo previous actions.
- **Queue** - Queue is a linear data structure in which elements can be inserted only from one side of the list called the **rear**, and the elements can be deleted only from on the other side called the **front**. The queue data structure follows the **FIFO** (First In First Out) principle, i.e. the element inserted at first in the list, is the first element to be removed from the list. The insertion of an element in a queue is called an **enqueue** operation and the deletion of an element is called a **dequeue** operation. the time complexity of enqueue and dequeue operations is also O(1) or constant time.
    - Types of Queue
        - **Linear Queue**: A linear queue is a simple queue in which the elements are stored in a linear manner, i.e., one after the other. It has a fixed size, and once it is full, no more elements can be added to it.
        - **Circular Queue**: A circular queue is similar to a linear queue, but the last element is connected to the first element, creating a circular structure. This allows the queue to reuse the empty spaces at the

front after the elements have been dequeued, making it more efficient than a linear queue.

- **Priority Queue**: A priority queue is a type of queue in which each element is associated with a priority value, and the elements are dequeued in order of their priority values, with higher priority elements dequeued first.
    - **Advantages**
        - **Provides a systematic way of processing data.**
        - **Useful in situations where processing data in a certain order is important.**
    - **Disadvantages**
        - **Limited capacity due to fixed size.**
        - **Overflows and underflows can occur if the queue is not properly managed.**
            - **Applications**
                - **Music player**: A music player may use a queue to manage the playlist, where songs are added to the back of the queue and played from the front.
                - **Web Servers**: Queues are used in web servers to manage incoming requests. When a request is received, it may be placed in a queue until a worker thread becomes available to process it.

- **Sorting** - Sorting is arranging a collection of items in a specific order. Efficient sorting algorithms can significantly improve the performance of these applications.
    - Bubble Sort - **O(n^2)**
    - Insertion Sort - **O(n^2)**
    - Selection Sort - **O(n^2)**
    - Quick Sort - **O(n log n)**
    - Merge Sort - **O(n log n)**

- **Hashtable**
    - Types of Hashtable
        - **Open addressing hashtable**: In open addressing hashtable, all elements are stored in the hash table itself. When a collision occurs, a new location is found by probing the table to find the next available slot.
        - **Separate chaining hashtable**: In a separate chaining hashtable, each bucket in the hashtable contains a linked list of key-value pairs. When a collision occurs, the new key-value pair is added to the linked list.
            - **Advantages**
                - **Fast access**: Hashtables provide fast access to data, with O(1) average case time complexity for operations such as search, insertion, and deletion.
                - **Scalability**: Hashtables can be easily scaled by increasing the size of the underlying array of buckets.
            - **Disadvantages**
                - **Hash collisions**: Hash collisions can occur when multiple keys are hashed to the same index, which can lead to decreased performance and increased memory usage due to the need for separate chaining or open addressing.
                - **Memory usage**: Hashtables can use a significant amount of memory to store both the keys and the values, especially if the load factor is high.
                    - Applications
                        - **Databases**: Hash tables are used in databases to index records and facilitate fast searching. Hash indexes are created on database columns to provide quick access to records.
                        - **Password storage**: Hashing is used to store passwords securely by applying a hash function to the password and storing the hashed result, rather than the plain text password.

**Hashtable collisions** - Hashtable collisions occur when two or more keys hash to the same index in a hashtable. In other words, collisions happen when the hash function produces the same index for multiple keys. When this occurs, the values associated with those keys can overwrite each other, resulting in data loss.

**Open hashing** - Open hashing, also known as separate chaining, involves storing all the key-value pairs that hash to the same index in a linked list. When a collision occurs, the new key-value pair is simply added to the end of the linked list.

**Closed hashing** - Closed hashing, also known as open addressing, involves finding a new location within the hash table when a collision occurs. There are several techniques for finding a new location, such as linear probing, quadratic probing, and double hashing. In closed hashing, the hash table itself stores all the key-value pairs.

- **Linear Probing**: In linear probing, the next available slot is found by checking the next index in the array. If that slot is occupied, the algorithm checks the next index until an empty slot is found.
- **Quadratic Probing**: In quadratic probing, the next available slot is found by checking the next index using a quadratic function. The function is of form f(i) = i^2, so the next index checked is (hash + f(i)) mod size.
- **Double Hashing**: In double hashing, the next available slot is found by hashing the key again and using that hash to compute the next index. The formula is of form f(i) = (hash1(key) + I * hash2(key)) mod size.